5-1-1997

Java based location independent desktop environment

Jeffrey Harman

JAVA BASED LOCATION INDEPENDENT DESKTOP ENVIRONMENT

by
Jeffrey I. Harman

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

at Rochester Institute of Technology

Approved by:

Dr. Andrew Kitchen, Professor, Computer Science, Thesis Committee Chairman

---

Dr. Muhammad E. Shaaban, Assistant Professor, Computer Engineering

---

Dr. Roy S. Czernikowski, Professor, Computer Engineering

May 1997

Java Based Location Independent Desktop Environment

I, Jeff Harman, hereby **grant** permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: May 19, 1997 Signature of Author: _____

ABSTRACT

JAVA BASED LOCATION INDEPENDANT DESKTOP ENVIRONMENT

by Jeffrey I. Harman

As the Internet grows and users become more mobile, users will need to be able to access from remote locations all of the same computing resources that they use in their offices. Some users accomplish this by taking everything with them, in the form of high-priced, high-powered laptop computers. This thesis investigates another possible solution to the problem of mobile users, that of a location independent user desktop.

This thesis and its related project will produce a JAVA based User Name Service, Application Service and User Desktop Application that use a combination of local and remote servers to provide location independent services. The Desktop application will provide a basic desktop environment to the user, including a tool bar and simple configuration management to demonstrate the basic principles of this system. The User Name Service will provide desktop configuration information to local and remote users. The Application service will provide (rudimentary) applications for the user to use on his/her desktop.

This thesis will investigate the basic properties of proxy services, configuration management, user key management, access control, and communication issues that arise in the investigation and implementation of a location independent environment.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to thank my wife Janine for her patience and support throughout this endeavor, as well Keith Miller whose technical expertise and writing skills have been indispensable.

GLOSSARY

API: Application Programming Interface.

AS - Application Service: A service supplying access to applications desired by the user.

Channel Key: An object that identifies an ongoing conversation between a service and its client. The Channel Key is issued once the Service Key has been authenticated by the service.

DNS - Domain Name Service: A standard naming convention used on the Internet.

Domain: An area of the Internet defined by a specific name.

FSS - File System Service: A service that provide access to a user's directories and files from both local and remote locations.

IUIS - Individual User Information Server: The individual thread in a UNS server that responds to a particular client's request(s).

IAS - Individual Application Server: The individual thread in a AS server that responds to a particular client's request(s).

JDK - Java Developer's Kit

LID - Location Independent Desktop: A desktop that is the same regardless of where it is accessed from on the network.

NFS - Network File System

Service Key: An object containing information about a specific service, and a license ID showing that the user has a right to use the service.

UNS - User Name Service: A service that supplies user authentication and account information.

Chapter 1

1. THE LOCATION INDEPENDENT DESKTOP

This chapter will describe the general concept of a Location Independent Desktop, and the software architecture used to implement it.

1.1. Background

The object of the Location Independent Desktop (LID) is to supply a user's desktop environments regardless of where he/she logs into the network; across the room, across town, across the country or around the world. The solution must support many types of computer systems, as the user may login into a different type of computer system from the one normally used.

The user's environment includes the look and feel of the user's desktop (color scheme, background, toolbar, etc.), applications, and access to the user's files. This scheme could be extended to cover access to other services (such as stream services for on-demand video, etc.), but are considered beyond the scope of this thesis. (This could be added in the future.) Also, this scheme assumes that the networks being used are secure. Therefore, the only security implemented will be that required for general user access to applications, accounts and files.

1.2. Software Architecture

This section will describe the software architecture of the system.

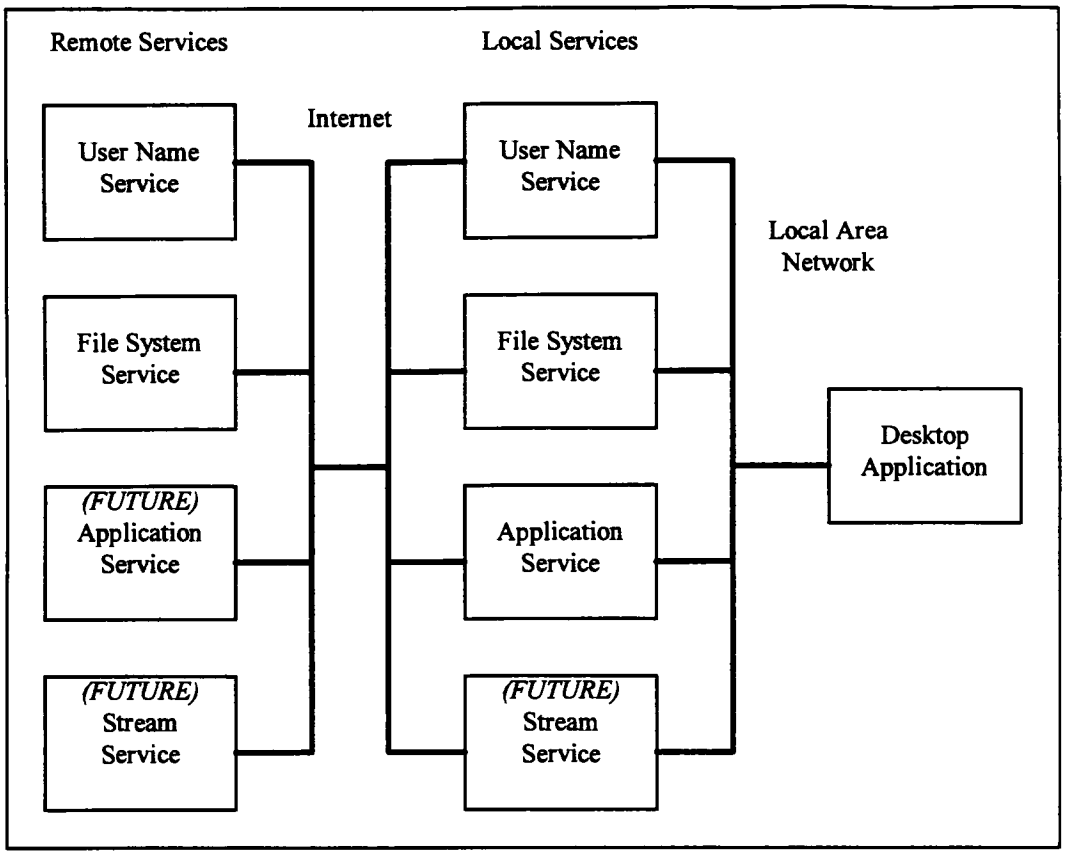


Figure I Software Architecture for the Location Transparent Desktop

The architecture can be viewed as three sections; the desktop, local services, and remote services. The Desktop Application provides users with access to their desktops, applications and files by contacting the local services. The local services determine whether they can meet the needs of the Desktop Application directly, or whether they must contact some remote service to fulfill the request. This is done automatically, without the need to inform the Desktop Application.

1.3. Desktop Application

The Desktop Application is responsible for providing the interface between the user and the services.

The user interface prompts the user for login name and password information (ex. MyName@rit.edu, MyPassword). Using the UNS API, the Desktop Application contacts the local User Name Service (UNS) to validate the user. If the user login name specifies another DNS address, the local UNS contacts the UNS at the specified address to validate the user. If the user is valid, the Desktop Application will query the UNS for account information such as the setup of the user's desktop, color scheme, etc., and use this information to display the user's desktop environment.

One of the standard applications that is provided to all users is a File Manager application. The File Manager uses the File System Service (FSS) to gain access to the user's directories and files, and display them to the user. The File Manager obtains a Service Key from the Desktop which allows it to use the FSS. The user can use the File Manager to traverse his/her directory tree, create directories, copy, move and rename files, etc.

1.4. User Name Service

The User Name Service (UNS) is responsible for maintaining the user accounts for a specific domain. It is expected that one of these services would be available in each domain where Location Independent Desktops are supported. When contacted by a Desktop Application, the UNS validates a user's login information, and supplies basic account information. If the user's account is at a remote site, the local UNS contacts the UNS at the remote site to complete any requests.

When the UNS receives a request it will supply a Service Key that authorizes the use of a particular service or application by a particular user The UNS will later be asked to authenticate this Service Key as part of the authentication process.

1.5. File System Service

The File System Service (FSS) is responsible for providing access to files and directories owned by the users of the system, and delivering them to the user's current location upon demand.

Authentication of the user is performed by receipt of a Service Key that was originally generated by the User Name Service (UNS). The FSS will contact the UNS to authenticate the Service Key, and then use the contents of the Service Key to identify the user and therefore determine his/her file access rights.

The local FSS will hide any accesses to remote FSS Servers that are required to fulfill the user's requests. When a remote FSS is being utilized, the local FSS will cache the directory and file data being used, and will attempt to anticipate what the user will need next, and obtain this information in advance so the user will not have to wait longer than necessary.

Sharing of files among multiple users will be allowed. A consistency method will be chosen, to keep multiple copies of the same file in sync.

1.6. Application Service

Application Service (AS) is responsible for providing access to applications and delivery of those applications to the desktop upon demand.

Authentication of the user is performed by request of a Service Key that was originally generated by the User Name Service (UNS). The AS will contact the UNS to authenticate the Service Key, and then use the contents of the Service Key to identify the user and determine the application access rights.

The AS will hide any unnecessary access information from the user in its pursuit of the application. In addition the application will have the ability to obtain information or files from its original provider using the same channel developed for itself. The application's original provider is the host that the application was taken from. This will give the application the ability to load libraries or other dynamic modules only when needed.

1.7. Key Paradigm for Services

The paradigm that is maintained by services in this environment is that of keyed service. There are two types of keys that are referenced in this environment: Service Keys and Channel Keys. Service Keys are used to obtain services and Channel Keys are used to access active services.

Service keys originate from the UNS. The UNS provides the Service key for a user as part of the hierarchy of application information. Each application and file system listed in the user's configuration is paired with a service key. The desktop simply decodes what kind of service is required by a user action, determines if the service has been initiated for that element. If the service has not been initiated then the service key is passed to the appropriate local server to initiate the service.

Service Keys are not simply a password for access to a service, they also contain information about where that service can be obtained if the local services are unable to assist the user, and information about the user. Service keys can be verified by the user's

primary UNS through the standard UNS API. These keys contain enough information to identify the user, password the service and the identity of the user's primary UNS provider for the FSS or the AS.

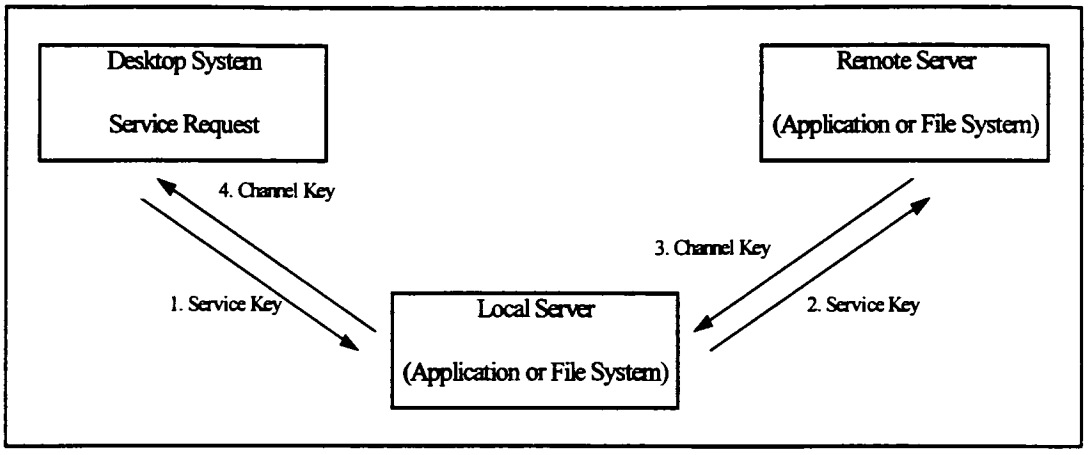


Figure II Service and Channel Key Information Flow

The general flow of keys is as follows: The user initiates an action (such as initial access to the file system or initial access to an application). The desktop then passes the Service Key that is associated with the activity to the appropriate local server. The local server examines the Service Key to determine if the request can be satisfied with local information. If the service cannot be satisfied locally then the Service Key is given further examination for information that indicates where the key may be used. The local server then contacts the remote server with the service key. The remote server should then respond with a Channel Key. The local server then sends a Channel Key to the user desktop which indicates that the service has been successfully initiated. The desktop can then request further information from the established channel without having to reinitialize the connection. Any failure within this chain will cause the local server to return an invalid channel key or a value that indicates failure (dependent on the

particular server interface or API) which will indicate that the attempted service request has failed.

Service Keys can be validated at two levels. First the service key can be verified by the UNS. The user's primary UNS server (which presumably provided all the service keys initially) can be queried as to whether the user is logged in and the key belongs to the user. It will respond with a message stating the validity of the key. Secondly, the end service provider (which presumably created the key initially) can check to see if the information provided by the key is valid. Only after both (or at least the second) checks have been made on the service key, is the channel key generated and returned to the user (desktop).

Channel Keys are only valid for the duration of a single activity (at minimum) or user login session (at maximum). Channel Keys are created at the inception of a service by the service provider (AS server, FSS server, UNS server, etc.) for the sole purpose of identifying the connection that has been established to service that request. Channel Keys are local access keys while Service Keys are full information keys.

As an example of the use of keys, consider how the File Manager and the File System Service (FSS) use keys to access the user's directories and files from a remote domain. Upon startup of the File Manager, the Desktop Application will pass the FSS Service Keys to the File Manager. (The FSS Service Keys were originally supplied to the Desktop Application by the local UNS, which in turn received them from the remote UNS.) The File Manager opens a dialog with the local File System Service using one of the FSS Service Keys. The local FSS contacts the remote FSS using the FSS Service Key to establish a dialog. The remote FSS will contact the remote UNS to authenticate the FSS Service Key. If valid, the service key is then examined locally by the local FSS to determine if it is valid for access to the service. If still valid, the remote FSS will generate

and return a Channel Key to the local FSS, which will, in turn, return it to the File Manager. The File Manager can then use the Channel Key with all further FSS API calls. The local and remote FSS Servers can authenticate the Channel Key themselves, without needing to contact the remote UNS.

1.8. Use of Java

The basis upon which the Location Independent Desktop is defined and implemented in Java. Java was selected for two reasons. First, this language was designed for the implementation of applications on the internet and, in particular, produces runtime portability on systems that support the Java environment. Secondly, programming in Java enabled us to gain experience with a new and important language and evaluate its strengths and weaknesses.

1.9. Network Domains

Rather than come up with a whole new naming convention, we decided to use and extend the DNS naming convention that is currently used in the Internet. This implies that each domain that supplies Location Independent Desktop Services must supply its own set of services (i.e. User Name Service, File System Service, etc.). These services may all run on one server machine, or may be run on separate machines.

1.10. Thesis work breakdown

Keeping the size of this project down to a workable amount was not an easy task. As has been mentioned, several areas were either left as "Future" functions, or were not addressed at all. We implemented prototypes to prove the major concepts described, and as a platform to explore their usefulness.

Jeff Harman concentrated on the Desktop Application, Application Service and the User Name Service. Keith Miller concentrated on the File System Service and the File Manager Application. Together they explored the concepts, developed the architecture and defined the APIs between the main modules. Once their respective areas had been developed and tested separately, they integrated them and experimented with the resulting system.

Separate theses were delivered, each concentrating on the areas defined for the author, with section one being the only common content. However, references that tie the documents together as a whole have been added where appropriate.

Chapter 2

2. BACKGROUND THEORY:

An examination of the design of this system reveals two primary differences between this system and other similar systems. The first is the dual nature (both primary and proxy) of the servers in this environment. The second is the separation of executables from the main user file system.

The dual nature of the system servers is a direct result of the differentiation between local and remote servers. When the remote servers are necessary to fulfill the clients needs, the local server becomes a proxy in all requests from the client to the server. The proxy nature or the servers is an important area of investigation.

An additional investigation of the design shows that the application system has been differentiated from the file system. Applications no longer reside within the user's file system(s). They are provided through a separate supply channel. The separation simplifies the job of the file system in that it no longer has to deal with the differentiation of data files and executable files, so the application system can concern itself with the issues of user verification and access privileges for application users only.

This separation of the file system from the application system is important division in the analysis of this design. The division represents an important departure from average system design in that most systems' executables reside within the main file structure. This division is intended to facilitate two important factors: caching differences and supplier separation. The caching differences arise mostly from the different access patterns for applications and for data files. The supplier separation is intended to allow vendors to supply applications to customers directly without having to provide the user

with the executables in some permanent media (such as CDROM or floppy disk). This allows vendors to manage versions and updates without having to directly supply clients with files and disks. This allows fully internet driven application provision and support. The file system then only supports data file access and related caching problems. The caching then becomes a "purer" problem for both systems.

2.1. Proxy Servers (Service)

The ability of local servers to become proxies for users is designed to simplify the desktop's job by providing a single point of access for all requests of a certain type. This simplification of the desktop complicates the servers in that they now have responsibilities as proxies for the user.

2.1.1. Proxy History

Proxy is defined as "A person authorized to act for another; agent or substitute" by The American Heritage Dictionary. Proxies have been used in the business arena for many years and have become accepted as an acceptable method of carrying out business. But being a proxy has always carried with it the burden of proof. This proof was often inferred from documents and signatures.

In the computer world proxies were first employed as a method of providing secure access to the internet for LANs that required insulation from the internet. These proxies acted for systems that were inside the LAN and were prevented from performing certain actions outside the LAN by firewalls or other security measures[Hardjono 93]. In particular World Wide Web access was often limited by firewalls and was facilitated through the use of proxies.

Eventually the use of proxies became quite wide spread. This spurred many groups to attempt to make the proxy servers faster, more reliable and able to perform a much wider range of functions [Shapiro 86]. The first major improvements were made by adding caching mechanisms to the server[Bestavros 95] to allow caching of commonly requested documents. This allowed common requests to be accelerated by eliminating the need to obtain the original information from the originator. In addition to providing WWW access to clients the proxies also began to become multifunction servers so as to allow FTP, Gofer, Archie, and other TCP based protocol services[Luotonen 94].

2.1.2. Strict Proxy Model

The strict model of a proxy (in a secure LAN) is that of a conduit for passing information through the firewall of the LAN. It is generally cacheless and only serves as a pass through and verifier of information. The information that passes through the server is examined and the destination and source are examined to determine if the information passes the particular security restriction within the LAN.

2.1.3. Popular Proxy Models

Proxies in common use today have evolved beyond the simple security role that they were originally developed for. They perform a wide variety of services such as subnet caching[Oguchi 96] and multiservice functionality in addition the basic tasks of security.

Subnet caching is where a group of proxy servers act in concert [Jeffrey 96] [Varadharajan 91] to perform more efficient caching of information. These systems often form hierarchies of servers that can interact with each other to request information that the others have already cached. This allows the groups of servers to more efficiently serve common requests to users. The patterns of interaction tend to either allow peer interaction in hierarchies or use some sophisticated communication and request system

to efficiently deliver the information to the users. This method of having interacting caching proxies could be easily applied to the servers in the LID to improve performance of the servers particularly in the area of application service, because the set of applications is limited and fairly static, so the common caching of applications would allow the hierarchy to cache applications more efficiently than the caching of the applications on one server.

2.1.4. Location Independent Desktop Proxy Model

The LID proxy servers do not follow any of the standard models for proxy service in that they also perform local server functions for particular set of users. This additional functionality requires a modification of the caching and security mechanisms employed to allow the system to properly serve client requests.

In this model of service the security issues become secondary to the problems of managing both local and remote services. The primary goal of a server of this kind is to act as a seamless server for all client requests. There should be no apparent difference for the client when the server is providing requests directly and when the server is acting as a proxy agent for requests.

2.2. Problem Verification and Discussion

The general paradigm shift in personal computing has been the move away from simple stand alone computing to much more net-centric computing [Yourdon 96] [Gellman 96] [Hamilton 96]. This shift has caused problems specifically in the area of applications that are not network friendly [Tomes 96]. This so called "fatware" is built to work well on stand alone stations but when it is run from a network file server it can severely degrade the performance of the LAN and the workstation. In some ways the structure of the

application is to blame, multiple DLLs, loadable data files and general inefficiency of application compilers cause applications to be very large. When a word processor takes in excess of 30 megabytes of disk storage, the simple act of transporting copies to multiple clients can be a detriment to LAN performance. In addition the move toward "office suites" and the inherent bundling of packages can cause even simple applications to become very large and unwieldy.

2.2.1. User Portability

Users have changed since the advent of the popular personal computer. Users have become much more mobile and demand that their software and data come with them [Tomes 96] [Hokimoto 96]. This is evidenced by laptop (and other portable system) sales and the move toward "telecommuting" that is occurring in the business place today.

Additionally, many business have multiple locations and multiple computing service provisions. These site and infrastructure differences (even within a single company) can cause many problems for professionals that move widely within the company. Couple these factors with a massive expansion in internet connectivity, it seems to leave a huge area of expansion open to development, that of a Location Independent Desktop.

This development would facilitate much more than business portability, it would allow home and office interoperability. This means that home and office access and software do not necessarily have to be split (they still could be). This would require that business servers must be security conscious in that they may have to only allow access to specific data from limited locations. But this is a logical extension of the existing server paradigm.

2.2.2. LAN vs. WAN Communication Issues

The server to proxy split parallels the different access paradigms. The local server tends to communicate with clients on a LAN, while proxy service tends to occur on a WAN. This split allows differences in access and packeting that facilitates optimization of caching and requests between servers and clients. Since LANs tend to be fast and reliable and WAN communication tends to be slow and unreliable, the differentiation of request becomes more valid. This also relieves the burden of attempting to optimize many different types of communication from and to the desktop. Servers can be optimized to perform the appropriate requests and access for the type of communication being performed.

2.2.3. Application Service vs. File Service Split

The initial inspiration for the split between applications and files systems came from the Harvard memory architecture employed on the Motorola 68040 chip. That architecture separated data memory from instruction memory. This allowed the caching to be optimized for each of the memory sections separately. This split is also justified by the acquisition of the different resources from independent sources and the requirements of applications that are net aware and make use of distributed resources [Dave 92]. This means that vendors could directly supply application(s) to the user without having to worry about the file systems that the data will be stored in. File systems also have different requirements than the servers that would provide applications to the user.

2.2.3.1. Caching Differences

One of the primary differences between applications and file systems is in caching. Caching of applications may be quite different from file systems. File systems need dynamically responsive caching mechanisms to react to a rapidly changing environment. Application caches can be much more static, only updating when the application has

changed (which is a much more rare occurrence than changes in data files) and because executables are read only. The only request that need be made to the originating application server would be to verify that that user has rights to run this application.

2.2.3.2. Security Differences

Applications and data files have different security requirements. Data files often contain sensitive corporate data, so the security must be flexible to allow limited access (to specific domains or users). While application tend to require use licenses. The difference in access require different security restrictions as well as server capability differences. An application should be accessible if the user has a valid access key while a data file may or may not be accessible depending on the requesting stations domain (location).

2.2.3.3. Communication Differences

Application access tends toward requests of a few files then little or no requests for more files, while a file system access tends to be very bursty and be required throughout the user's session. These differences would require different communications systems to accomplish efficiently.

2.2.3.4. Maintenance Issues

Application Maintenance is very different from file system maintenance. For the developer of an application updating or changing an application can be a huge problem specifically because distributing the fix to all of the clients that use the system is difficult. This system would allow the simple changing and updating of application because the developer would have a limited number of application servers that contain the original application that is requested by users of the LID system.

File system maintenance becomes much more simple when application service is separate. There is no longer a need to worry about applications and execution privileges.

This allows system administrators to concentrate on the complex issues of data base management and simplified file management. Also, file system administrators can limit the security measures to read and write privileges for users of the system.

2.3. User Configuration and Maintenance

One of the goals of this architecture is to consolidate the problems of user control and configuration into one system [Youxun 96]. This will allow user's environments to be portable in that they can obtain all their setup information from one source. This source would allow the support of multiple types of desktop through the homogenization of some of the data and the ability to store information that is specific to a particular desktop. (note: this particular project will not implement the latter but it is a logical extension of the project.)

Once all the primary information about a user is centralized (for a particular set of users) it allows the user the freedom to move between machines without detriment because each machine would appear to the user as his/her own machine because it implements the configuration that it obtains from the user's server. This system independence will allow users to no longer be paired to a specific machine.

2.4. Architecture Validation

This section will endeavor to present some of the pros and cons of this particular architecture. Much of the acceptance or rejection of the system will be made by users and system administrators based of perceived need and cost to install. But, the benefits of this system (centralizing maintenance, user portability, and developer simplification) could make this system viable and accepted.

2.4.1. Combination Proxy/Server Model

The combination of pure server and pure proxy servers into the combination server will allow the servers to become more responsive to user requests. The improved responsiveness will be a result of the improved caching nature of the servers and the proxy nature allows the optimization of communication for LAN independent of WAN optimizations.

2.4.2. LAN / WAN Seamlessness

The seamless nature of the desktop's access to LAN and WAN resources (facilitated by the dual server nature) allows the design of the desktop to be simpler. It need only know about the location of its servers and maintain the user's state. User control becomes the primary mission of the desktop while the communication issues become much more simplified (especially with the use of service APIs). Security concerns are also greatly reduced for the desktop as most of the burden of verification is offloaded to the servers.

In addition to the apparent seamlessness of the LAN/WAN access to the user, the system administrator can user the local to remote division as a simple point of restriction for security measures. The WAN access side can be limited to only the transactions of an allowable type, and thus allows the entire sub-network to be shielded from hostile actions. This has always been a problem for system administrators and has necessitated the invention and widespread use of firewalls because the administrators could never verify that all system on the network were secure from attack from user software

2.4.3. Speed Detriment

The flexibility of this environment does not come without penalty. The necessity of extra interface layers on both the desktop through various APIs and the servers (to facilitate seamless LAN/WAN access) will slow the access to services somewhat. In

addition the addition of a proxy server into the communication pipe between a server and the desktop (no matter how well cached) will degrade performance somewhat. But, the API layer may be inherent to many designs so the speed detriment associated with an API is somewhat alleviated in a comparison to other environments.

This architecture will also be very sensitive to server loading. If the servers are overloaded with users the performance for most (if not all) users could be degraded. The addition of subnet caching and other proxy caching mechanisms could allow the alleviation of this problem somewhat but will never eliminate it.

2.4.4. Speed Enhancement

The caching mechanisms (particularly of applications) could greatly improve access to remote servers over direct request of information from the desktop to the remote server(s). The locality of the information and the different access patterns from LAN to WAN could greatly improve performance of remote server access. In addition the protocols could be optimized to perform well at the two levels (LAN and WAN). This alone could greatly improve performance of the retrieval portions of the system.

In addition, the offloading of work from the desktop to the server could free the desktop for more computing power to be applied to the problems of graphical display or other issues that could be detrimentally affected by the requirement that the local system perform many of the functions that are taken up by the servers.

2.4.5. Maintenance Difficulties

The storage of user configuration information in a single site could cause a problem in that different versions of the desktop environment may interpret the configuration information differently. This could cause problems as users become very mobile and use

a variety of manufacturer platforms. This problem could be lessened (but probably not eliminated) by a very well documented specification of the methods and intent of all the user configuration features.

2.4.6. Maintenance Simplification

Desktop configuration is vastly simplified because individual machines need no longer be configured in any way except to let them know who their local servers are. This could even be automated by a mechanism such as the Portmapper from RPC™.

Users need no longer worry about location and format of information. As long as they can get to a machine that runs the LID then they can access the appropriate application and view data in its native application.

The enforcement of firewall policies and systems becomes much easier. The firewall is pierced only by the servers (local machines cannot access the internet directly) and the servers (in the future) should be able to enforce access restrictions for requests originating within the LAN and from WAN sources.

Additionally the use of Java makes application development simpler in that the final executable is not tied to a particular machine but is instead run on the Java virtual machine. The virtual machine can be built on any platform and has been implemented on most of the popular computing systems. This generalization means that systems will no longer have to be reworked for every new platform that is wanted.

Chapter 3

3. KEYS

3.1. Overview:

The discussion of keys is an important factor in determining the viability of this system. Keys are intended to perform multiple tasks in the system. The tasks range from identifying users, identifying user requests, providing a mechanism for the transportation of (possibly secure) information, and verification of communication.

These many duties require that the structure of the key be flexible and still support common use and information containment. What has been left out of the key (in terms of fixed information) is just as important as the explicit information contained within the key.

There are two types of keys: service keys and channel Keys. Service keys are used to start services and are used by the system as a token of rights, privileges and identification. Channel keys are used to maintain and help verify the identity of systems as they communicate between themselves.

3.2. Motivation for Key System Creation

The motivation for the inception and design of the key system is the simplification and unification of the management system for user resources. For this system to work the environment must be able to simply determine what requests are being made of it and respond accordingly. The key system allows the simple management of a single data structure that contains all the necessary information to start a new service. This is the

essence of the service key. The key is flexible and allows the implantation of any information necessary to start the service. Primarily, the keys are used in either file system service or in application service but the key system (and associated managers) could be expanded to handle any new service(s).

The channel keys have a different use. They are intended to perform the function of a communication token for inter-system communication. The channel key is intended to allow the systems involved in communication to establish the identity of both parties in the communication (through both pieces of information contained within the channel key) and as such is passed at the inception of each new request or reply. Information contained within the key could be used to perform extra security functions such as public key encryption or other security measures (through the use of the byte field).

3.3. Service Keys

The service key contains three main blocks of information. The first block is user identification, the second is request information and the final is the byte key. These blocks comprise the primary source of information about the user and the user's request.

The user information block consists of three main pieces of information. The first is the user's name. It is the same name that the user uses to log in with (except the DNS host name). The second is the user's primary UNS host, this is the host that is concatenated with the user's name to form the login name. Together these two pieces should form a unique identifier for the user. The third is the UNS host port number. This allows the UNS host to use a port other than the standard known port for access.

The request information block contains two pieces of information. The first is the request field. This field contains information about the request that the key is intended

to allow access to. The second is the display field. This contains information that is displayed to the user in list selection choices.

The byte field is an unformatted byte array of arbitrary size. This was intentionally left unformatted to allow different services (that require different information) to utilize the byte field to include more information.

The structure of the service key is detailed in Figure III:

Service Key Data Structure

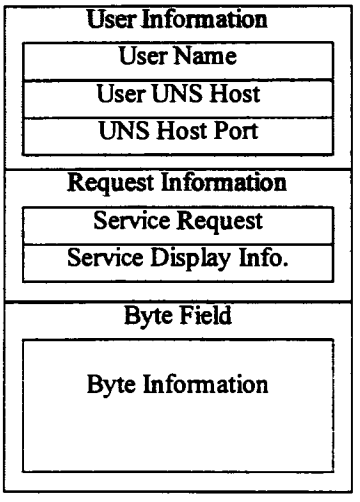


Figure III: Service Key Data Structure

3.3.1. Service Key Data Storage

The data storage for the service key is not restricted by size but by use. The fields (except the byte field) are stored as strings in Java (or as an integer for the UNS host port). Java strings have no upper limit (except the memory of the system) so there is no real restriction on size. Practically speaking, the upper limit is set by the displayability of the

information in normal contexts. The user name and user UNS host is restricted more by the display and entry system (about 80 characters) than by the storage and transmission mechanisms. The port is an integer so its storage is fixed. The service request is limited by the restrictions of the Java string class. The service display information is limited to 128 characters (for display purposes). The byte field is unlimited (see the discussion of the byte field later in this chapter for a more in depth discussion).

Since the stored fields are objects within Java, maintenance variables that contain information such as size, are unnecessary. In addition, the data structure contains no direct information about the limitations on the size of the fields. This is not necessary because the communication channels in Java allow for the transmission of all basic data types through the user of specialized input classes. This allows the transmission of Strings and Integers without any special handling. The transmission of byte arrays requires the sending of the length of the array before the array but is not limited in any other way. These mechanisms allow the structure to be very flexible and only limited by display and realistic transmission parameters. This allows the maximum flexibility of the key data structure. Channel keys are limited by the same constraints.

3.4. Channel Keys

The channel key contains two blocks of information. The first is the user identifier and the second is the byte field. The channel key is a token that indicates that the communication that is taking place can be verified and is being carried out by the appropriate agents.

The user identifier field contains information pertaining to the identity of the user. The identity that is placed in the field is intended to allow the easy identification of the key to verification agents.

The byte field is the same as the byte field in the service key and contains an unformatted block of bytes of arbitrary size.

The structure of the channel key is detailed in Figure III:

Channel Key Data Structure

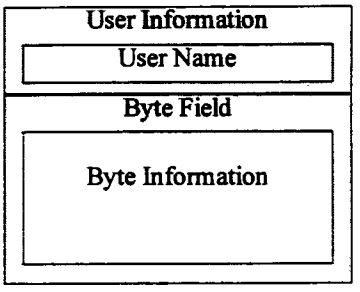


Figure IV: Channel Key Data Structure

3.4.1. Channel Key Data Storage

As with the service key the storage is limited mostly by the types of data that contain the information. The User name is stored as a Java string and there is no restriction on its size. The byte field is set up intentionally not sized to allow for maximum flexibility of the data structure.

3.5. Byte Field

The byte field in both key designs allows the particular agents and services to customize the key to contain more information. This information can be of any sort (as Java allows the simple conversion of most data types to byte arrays via a piping mechanism). In addition, the byte field itself can be used as a verification mechanism with the use of simple security checking mechanisms such as checksums or rotation of bytes.

What has not been addressed in this design (specifically) are the issues of more sophisticated security mechanisms. Mechanisms such as public/private keys and other encryption systems are not inherent to the key system. The key system, however, could be easily adapted to such systems with the appropriate embedding of encryption information and the encryption of information in the byte field of the keys.

Chapter 4

4. PROJECT DESIGN:

4.1. Overview:

The design of a location independent desktop is concerned primarily with the issue of transparency. . It is critical that the user be unaware of the mechanics and logistics of delivering his/her resources to the desktop if this system is to be successful. This is because if the users feel that they have to keep track of WAN resources they may feel that the system is too complicated or requires too much knowledge about computing systems. The seamless nature of the system shields users (particularly non-technical users) from the actual nature of the system. The users should be unaware of the mechanisms employed to verify rights unless the attempt to use a particular resource has failed.

The scope of this project precludes the construction of a fully functional desktop operating system. The desktop system will rely on the native operating system of the platform(s) on which it resides for resource management beyond the resources that are required to carry out the system tasks of the desktop and server systems.

In addition, the servers will be basic applications that reside on different hosts. They will not attempt to become actual operating system servers or daemons that are registered or are independent of user login/logout.

4.2. Major Elements:

This project will entail the construction and management of three separate entities: Those entities consist of a User Desktop, User Name Service Server(s), and Application Service Server(s). The user desktop will be a simple application with the sole purpose of allowing the user to: spawn applications, login, logout, modify simple environment settings (colors), and add or subtract elements from the user Hierarchy. The User Name server will manage the configuration, authentication activity, and remote user actions. The application server will manage the initial access to applications and the primary application executables.

These elements should work in conjunction with the File System Service (provided by Keith Miller) to provide a user with a functional location independent desktop environment. Each element will be detailed in this architectural document. This overall system should appear to the user as if all of the resources being accessed are local, providing operation transparency regardless of location.

4.3. User Desktop:

The primary user interface to this system will consist of a simple pane with two buttons: User Button and System Button. The User button controls the access to the user for actions that are available both before and after login. These actions include configuration changes, Login/Logout, viewing of file systems and spawning of new applications. The System button controls system level actions including; addition and modification of hierarchy elements for both applications and file systems, orderly system shutdown, system status and any maintenance functions that the system may require.

A click of a button will bring up a list of choices for the user to view. Any actions that are disabled will not appear on the list. The user (upon selection of a particular element will be generally be provided with a window prompting further input. These will be tailored to the specific needs of the user action.

The user desktop consists of ten primary elements: The User Interactive Controls, Desktop Manager, Application Manager, File System Manager, Configuration Manager, User Name Service API, Application Service API, Execution Manager, Execution API and Login/Logout Manager.

The User Interactive Controls will provide the graphical user interface that the user is presented with. It will be able to draw and dismiss any window or menu that the system requires to interact with the user. This object will be instantiated once for the duration of the desktop's existence.

The Desktop Manager provides support for actions that are initiated by the user through the User Interface. It will gather information from resources (remote and local) though other temporary objects. The Desktop Manager will instantiate objects as required to carry out user requests and user information. The desktop also acts as a communication conduit between applications and user information. The Desktop Manager is instantiated once for the duration of the desktop's existence.

The Login/Logout manager is responsible for the maintenance of user sessions. This entails service initiation and service termination. It will be instantiated to service a user login and will be terminated at user logout (or system shutdown if the user is logged in when the system is shut down). It will complete its tasks through the use of the UNS API.

The Application Manager is responsible for the maintenance, information dissemination to and from the user hierarchy of applications, and service key management for elements in the user application hierarchy. It will be instantiated after a successful user login and destroyed after a user logout. It will be able to store and retrieve Application hierarchy information through the use of the UNS API.

The File System Manager is very similar to the Application manager in that it maintains a list of service keys in a hierarchy. In this case the keys only refer to the file system(s) that user has access to. It is instantiated after login and destroyed after logout. It uses the UNS API for information retrieval and storage.

The Configuration Manager, like the File system manager and the Application manager manages some user information, in this case the user's desktop configuration. It is also created after a successful login, destroyed after logout and uses the UNS API for the maintenance of its data.

The Execution Manager is responsible for the creation and communication to applications that the user requests to be started. It will be instantiated by the desktop for each new application and is destroyed by either a user logout or the applications closure. It will obtain the initial application through the AS API.

The EXE API allows executables (processes) to communicate with the desktop's environment so that it can make requests (such as FSS service keys). It also allows access to the text output (System.out) of the desktop.

The UNS API is responsible for all communications to the local User Name Service server. It will act as a conduit for all requests and updates that the desktop requires for its

functionality. It will maintain the connection to the UNS server. It is instantiated by the Login/Logout Manager to service a login and is destroyed after a logout.

The AS API is responsible for communications to the local AS server. It is instantiated by the Application Spawner and is only maintained until the application has begun operation. Further needs of the application (in terms of modules and non user loadable files) will be handled by the FSS. The general outline of the object inter-relationships is shown in Figure V.

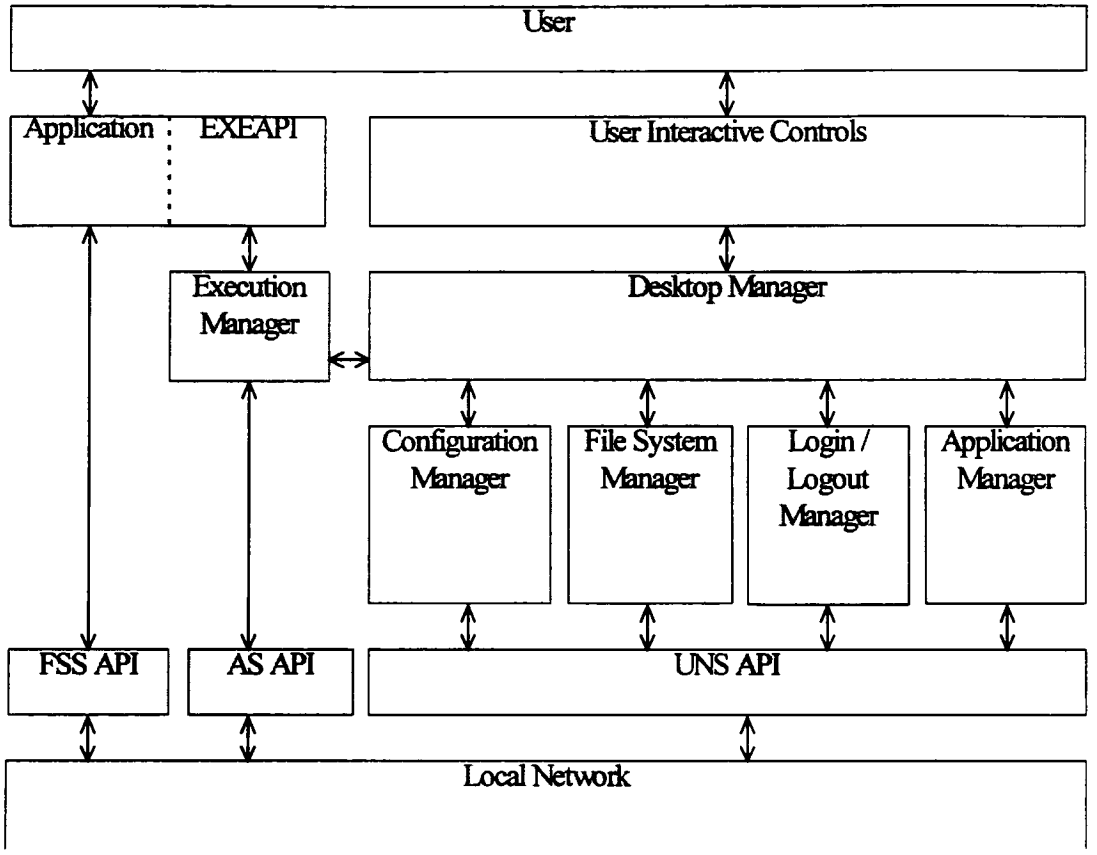


Figure V Desktop Application Major Object Interface Layout

4.4. User Name Service Server

The user name service server is responsible for providing user information for users that are directly supported by this server and indirect support for users who have remote UNS providers. The UNS maintains several different main elements to accomplish its task of providing service. They are: The UNS Communication Manager, Parser, Mediator, User Information Database and Individual User Information Server(s).

The primary job of the UNS server is to provide three basic services. The first basic service is to provide information to locally registered users. The second basic job is to provide an information conduit for users who are on the server's local network but have remote UNS providers. The third service is the verification of service keys for local users who maintain their accounts on this server.

There is no external interface provided for the UNS Server. The server will display system activities to the system prompt. This display will consist of all major user transactions (login and logout) for inspection and verification purposes. The display will be shown at the text prompt that the UNS Server was started from.

The UNS Communication Manager is the object that is in charge of the initiation of new service requests. This means that it listens on a fixed port (a port that has a well known value for all UNS Servers) and waits for new communications to be initiated. Once an incoming message is detected, a new port is spawned to service it and the port is passed to a Individual User Name Server for handling. This object is instantiated for the lifetime of the server.

The Mediator is responsible for internal communication between the User Information Database and Individual User Information Servers. It will be the primary method of information retrieval for the Individual User Information Server and the method by

which keys will be verified by active accounts. It is instantiated for the lifetime of the server.

The User Information Database is responsible for the dissemination and update of user information to users that have local accounts. It uses the Mediator to communicate to all its clients as well as to receive information requests and updates. It is instantiated for the lifetime of the server.

The Individual User Information Server is the communication conduit for the dissemination and update of user information (including application and file system hierarchies and configuration). There are three basic functions of this particular object. The first function is for users who have accounts on this machine. The second is for users (on the local LAN) who are using this server as a conduit for requests to remote UNS server(s). The third use is for key verification. The first use IUISs use the local communication medium (the mediator) to make requests to the local User Information Database to fulfill requests and updates. In the second case, IUISs simply pass all requests and replies to the remote UNS or local user for action. In the third case, IUISs verify the service key that is passed to them then close the connection. The Individual User Information Server is instantiated for the duration of the user Login session of the desktop or key verification request. At startup the IUIS determines which mode of operation is necessary for the current session and appropriately sets its mode of operation. Upon termination of the Login session on the desktop the object is destroyed.

The major object connections are outlined in Figure VI.

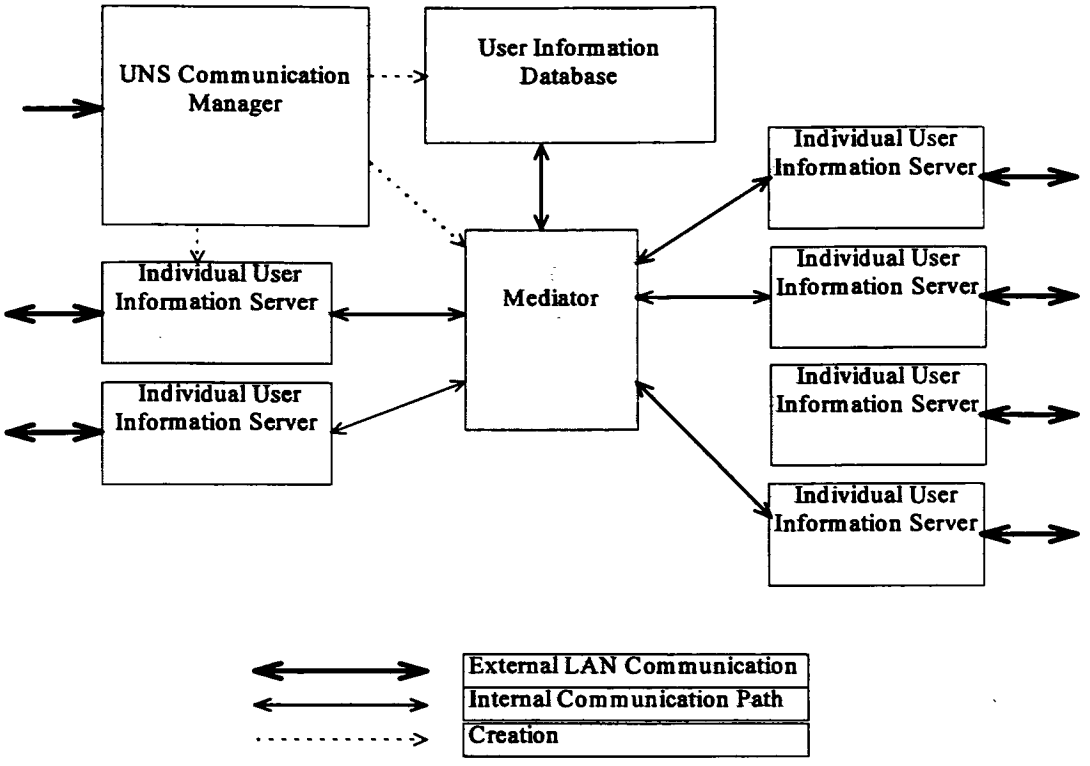


Figure VI UNS Server Major Object Interface Layout

4.5. Application Service Server:

The application service server is quite simple. It consists of only three elements: The AS Communication Manager, Individual Application Server (IAS) and the UNS API. It is responsible for the transmission of new application to the desktop.

It has three basic jobs. The first job is to verify that the service key provided is valid (through the UNS API) and correct (i.e. contains the appropriate user identification to access the application). The second job is to provide the requesting desktop (or remote

AS) the initial application executable to spawn. The third job is to create a communication conduit for local desktops to communicate with remote AS servers.

The AS Communication Manager is responsible for the initiation of new service requests. When a new communication is detected (on a well known port) it passes the request (via a new port) to an IAS. It is instantaited for the lifetime of this server.

The Individual Application Server is responsible for the inception, verification and servicing of single application requests from the desktop. The IAS uses the UNS API to verify the validity of the service key. Then, it uses the key to determine which application is being requested and, when requested to do so, it provides the basic application to the desktop. It is instantiated for the duration of the application service request.

The UNS API is responsible for the verification of service keys through the appropriate UNS server. It is a simple pathway for calls to be automated for the verification process. It is instantiated by the IAS for the verification then is destroyed.

The major object connections are outlined in Figure VII.

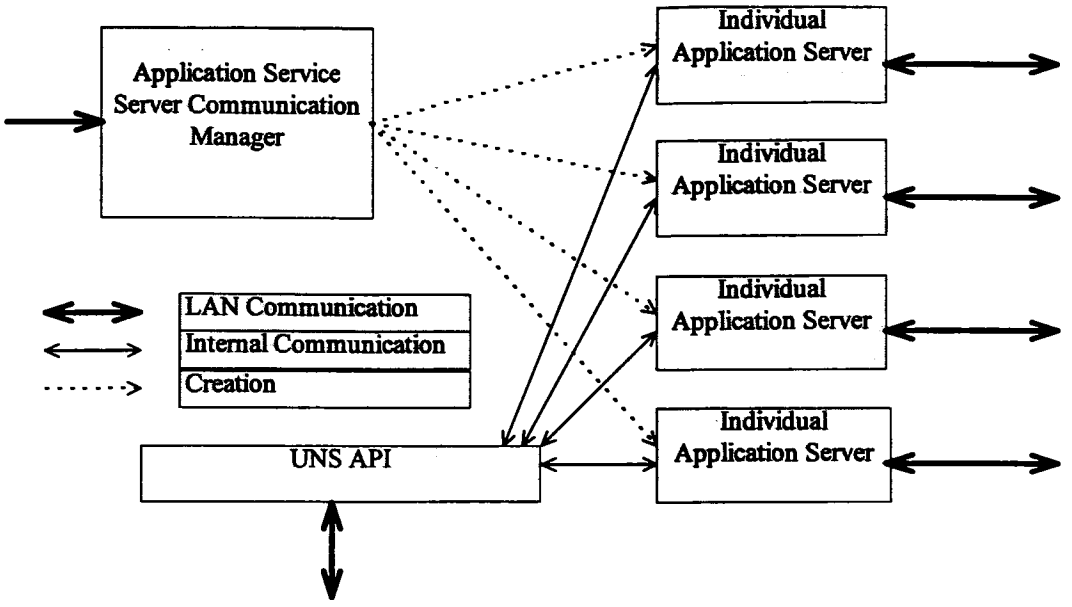


Figure VII AS Server Major Object Interface Layout

4.6. Use Cases:

The use cases will attempt to outline the basic flow of operations to carry out basic system operations. The four basic use cases are: local user login, remote user login user logout, and application inception. These basic operations give a general picture of information flow and system activities.

4.6.1. Local User Login:

A local user login occurs when a user attempts to log in to the desktop and the local UNS server has information about the user. This means that the local UNS server can validate the user directly and does not have to use a remote server to verify the user.

1. User Selects System Button from Desktop console and then selects login from available selections..

2. User provides user name (including domain information i.e. jones@uns.spam.edu) and his/her password.

3. Desktop Manager creates a Login/Logout Manager and provides it with the user information.

4. Login/Logout Manager creates a UNS API and requests a new user login.

5. The UNS API contacts Local UNS Server and initiates new service.

6. The UNS API provides the new Individual User Name Service Server with the user name and password.

7. The IUNS uses the information to determine if the user is local. If the user is local then it polls the User Database for the validity of the user's password. If the password is OK then the IUNS replies to the UNS API that the user is valid. If not the IUNS replies that the user is invalid, then terminates.

8. If the IUNS validates the password, the Login/Logout Manager replies to the Desktop Manager that the operation was successful and supplies the Desktop Manager with the handle to the UNS API. If the IUNS invalidates the password, the Login/Logout Manager destroys the UNS API and then Replies to the Desktop Manager of the failure. The desktop manager would then destroy the Login/Logout Manager.

9. If the login was successful, the Desktop Manager creates both application and file system managers and a configuration manager and provides both of them with the handle to the UNS API.

10. The Application Manager requests the user application hierarchy from the UNS API.

11. The File System Manager requests the user file system hierarchy from the UNS API.

12. The Configuration Manager requests the user configuration from the UNS API.

13. Once the Configuration Manager has successfully obtained the user configuration, the Desktop Manager requests the information so that it can display the desktop appropriately to the user.

14. The Desktop is updated with the user's configuration and the system waits for the next user interaction.

4.6.2. Remote User Login:

A remote user login occurs when a user attempts to log into the desktop and the local UNS server is unable to verify the user because the user's information is not stored in it. The local UNS server then becomes a proxy for the user and forwards all further requests and the current login validation to the appropriate remote server.

note: Steps 1 through 6 are the same as the Local User Login.

7. The IUNS uses the information to determine if the user is local. If the user is not local, it creates a conduit for connection to a remote UNS. It then attempts to contact the remote UNS and validate the user information.

8. The Remote UNS server either validates the user or not and then replies to the local UNS as if it were the requesting desktop. If the user is validated by the Remote USN then the IUNS passes the information onto the desktop and configures itself to act as a communication conduit for future requests. If the remote UNS invalidates the user, the invalidation is passed onto the UNS API and the IUNS destroys itself.

Steps 9 through 15 are the same as the local user login steps 8 through 14.

4.6.3. User Logout:

User logout attempts to log the user out of the current UNS server (either remote or local) and terminate the channel that has been created to supply the user's information. If the user is logged in remotely then the local UNS server will relay requests to the user's remote UNS server.

1. The user selects the system button from the user console and selects logout from the information displayed.

2. The Desktop Manager sends a close message to the Configuration Manager and the Hierarchy Manager. The Configuration manager determines if any changes where made during this session. If so it sends updates to the UNS API to be saved on the user's UNS server.

3. The desktop instructs the Login/Logout manager to logout out of the UNS.

4. The Login/Logout Manager sends a logout to the UNS API.

5. The UNS API sends the logout message to the local UNS server (IUNS server).

6. The IUNS rebroadcasts the message if the user is remote, if local it closes all open connections with the mediator and the data base manager, responds to the UNS API that it is terminating the connection and destroys itself (and the connection). If the user is remote it rebroadcasts the message to the remote IUNS and when the closure reply comes, it destroys itself after rebroadcasting the message to the UNS API.

7. The UNS API sends the successful logout message to the Login/Logout Manager.

8. The Login/Logout Manager sends the logout message to the desktop then destroys the UNS API.

9. The Desktop Manager then destroys the Login/Logout Manager, Hierarchy Manager, and the Configuration Manager.

4.6.4. Application Inception:

This is a general outline of the starting of a new application on the desktop. It requires that action has been requested by the user and the appropriate service key for the user's application is available.

1. The user selects the user button from the user console and selects on of the applications presented.

2. The Desktop Manager Creates an Execution Manager Thread and uses Java's Process mechanisms to capture the child's (executable's) input and output streams to provide a communication conduit to the child process.

3. The Execution Manager requests the Service key for the application from the Desktop Manager who in turn requests it from the Hierarchy Manager.

4. The Execution Manager then creates an AS API. It then provides the AS API with the service key and instructs it to initiate service.

5. The AS API contacts the local AS API with the service key.

6. The AS server then spawns an IAS server to handle the request.

7. The AS API passes the service key to the IAS server and requests the creation of service.

8. The local IAS server examines the key and decides if it is locally serviceable, if not determines what AS server can and contacts it with the service key for the inception of service. If it is locally serviceable, it tries to validate the service key with the information provided in the key.

9. To validate the key, the IAS server first creates an UNS API and contacts the user's UNS server with the service key. If the UNS server responds that the key is valid then the key is further validated for content and access privilege.

10. If the key is validated a successful message is passed onto the calling machine and then IAS server is maintained. If it fails then the IAS server sends a failure message to the AS API.

11. If the Key is validated and the service channel created, the Application Execution Manager then requests the application from the AS API. It is stored locally until completed. Then the application's Execution Manager starts the application.

12. The Application can then use the EXE API to contact the desktop to request the list of the user's FSS service keys. It can then use the keys to initiate file services with the FSS API.

Chapter 5

5. PROJECT IMPLEMENTATION DISCUSSION

5.1. Packages and Utilities

The Java programming language uses a feature called a package to group objects of a similar nature (functional or intent). In this project, packages are used to create sets of objects that implement a particular element in the design (such as the desktop) or to create sets of utility objects that are used in many parts of the system. In addition to the utility objects, APIs are also grouped with objects that are necessary for their functionality.

The general packages are described in the following subsections.

5.1.1. Keys

The Keys package contains the basic key objects (service and channel) and objects that are related to their generation and use. The uses of service and channel keys have been outlined in previous chapters.

5.1.2. UNSUtils

The UNSUtils package contains the UNS_API and all the custom objects that are passed into and out of it. The UNS_API contains all the necessary functions to communicate with the UNS Server from a client. The Hierarchy Manager contains functions for listing and managing service keys. The Configuration Manager contains all the desktop setup information for a single user.

5.1.3. ASUtils and DesktopUtils

The ASUtils package contains the AS_API. The AS_API contains all the necessary functions to communicate with the AS Server from a client. The DesktopUtils package contains the EXE_API. The EXE_API contains all the necessary functions for communication with the desktop from a spawned client.

5.1.4. ASServer, Desktop and UNSServer

The ASServer, Desktop and UNSServer packages contain all the files for functional AS Server, Desktop and UNS Server except the ASServer, UNSServer and Desktop objects (that reside one directory level higher). The objects contained within the packages are intended to constitute the complete desktop and servers. The ASServer, UNSServer and Desktop objects contains the main routines that actually run the servers and the Desktop as Java applications.

5.1.5. Miscellaneous

In addition to the packages outlined above the system provides a TextEditor object that demonstrates the use of the EXE_API and is a simple application that is executable through the AS Server.

5.2. Managers

The managers (Configuration, Application, File System and Login) that appear in the design as a single object are actually implemented as several objects. The extra objects generally provide interfaces that allow further user interaction.

5.2.1. Configuration Manager

Figure VIII shows the general structure of the Configuration Manager.

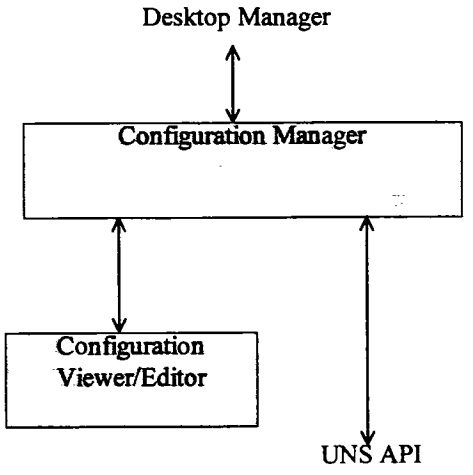


Figure VIII: Configuration Manager

The UNS API and the Desktop Manager are similar to the general design objects. The Viewer/Editor has been added to provide a convenient method for the user to view and edit the current configuration.

5.2.2. File System / Application Manger(s)

The File System Manager and the Application Manager were both expanded to allow multiple operations to be performed on the list of service keys associated with each group. The general operations were: Viewing (and selecting) and modifying (including selection for modifcation).

Figure IX shows the general structure of the File System and Application Manager(s):

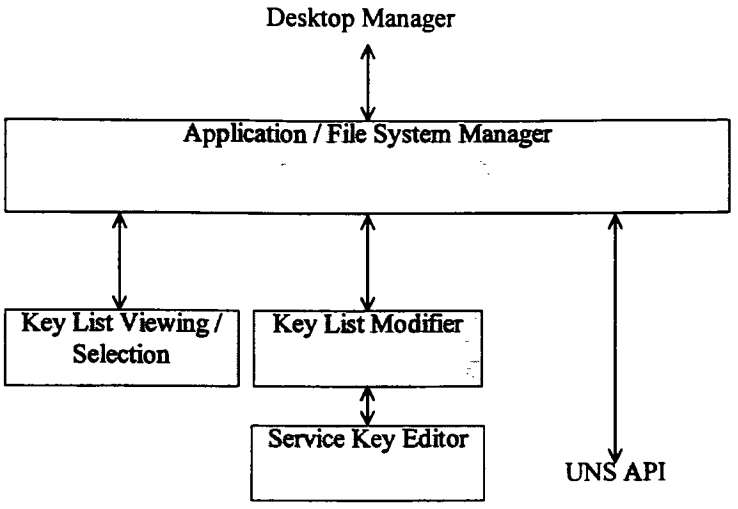


Figure IX: Application/File System Manger(s)

The UNS API and the Desktop Manager are similar to the general design objects. The Key List Viewing and Selection object has been added to facilitate selection of keys to be listed (viewing of keys for File System Manager and Selection of Applications for Application Manager). The Key List Modifier was added to display the list of current keys and allow the viewer to select a key to be modified or add a new key to the list. The service key editor is used to create a new key or modify an old key.

Chapter 6

6. PROJECT EVALUATION

This chapter evaluates this project. Functionality is the primary object of the evaluation. Performance metrics were not essential as the project is intended to provide a means of demonstrating the feasibility of this project and provide a test bed for future enhancement and modifications. As such, the performance (beyond a crude qualitative overview) is not discussed at length.

6.1. Functionality

6.1.1. Desktop

The Desktop prototype created as a part of this thesis was intended as a "Proof of Concept" vehicle to demonstrate the viability of this interface and to show that Java was an adequate language for the development of such a system. As such it succeeded, although there are reservations which will be documented here. The resulting prototype meets all the functions intended, including platform portability (some GUI elements do not appear perfectly on some platforms), user interaction and application spawning. The application spawning process was originally intended to be of a thread nature, but the restrictions of the runtime Java environment necessitated the conversion of the subthreads to full processes.

The conversion of applications from threads to full processes was necessitated by the inability of the Java language to spawn threads of an unknown type. By unknown type, it is inferred that the spawner is unaware of the exact type of the class to be started as a thread. This means that if (at compile time) the Desktop system is unaware of any possible application that it might be requested to run it would be unable to run the

unknown application. The full process system allowed the application to be downloaded (as class files) and started without foreknowledge of the classes that are necessary to run the application. There may exist a solution for this problem but it is at best a non-trivial problem.

The EXE_API provided a reasonably simple interface between the child processes (applications) and the desktop environment. But, it necessitated the restriction of System.in and System out (equivalent to UNIX standard in and standard out) to calls governed by the EXE_API.

The restriction on standard in and standard out for applications was necessitated by the inability of Java to easily pass a pipe or other communication conduit to the child process. The child process is started in a completely separate environment so sharing of data or objects is allowed through normal intra-application channels. The Java process calls did (however) provide a communication conduit through the system in and out of the child process. The parent process has complete control of the child's system in and out. This project takes advantage of this and creates a communication conduit through the child's system in and out.

The Java AWT (Abstract Window Toolkit) turned out to be somewhat of a disappointment. The window management (placement of graphical objects) system was very difficult to make work properly, and, when it was transported to different platforms, did not behave in a consistent manner. In particular the sizing of objects varied greatly from platform to platform. Perhaps the next version of the AWT in Java 1.1 will cure some of these difficulties, but that remains to be seen. Also, the color management of the windows did not seem to work at all, in that no method could be found to modify the basic color scheme of the windows. It is assumed that there is a

working method for changing the colors of windows, but at this time the author has been unable to determine the appropriate method.

6.1.2. UNS Server

As with the Desktop, the UNS Server prototype created as a vehicle to show the feasibility of this interface and that Java was a reasonable choice of language and environment. The project succeeded in demonstrating this point, because it carried out all the required functions without any major speed or complexity problems. The server included all the necessary functions to implement this design, including user login, user logout, key management, key verification and configuration management.

The UNS API was also very successful in providing a simple interface between clients and UNS Servers. The API proved to be successful in verifying keys for the FSS Server as well as providing services to the prototype desktop. For a detailed description of the UNS API please refer to "Appendix C: Programmers Guide".

The multi-threaded server design was successful in its intent of providing both proxy and direct support for client requests. In this version the information database was very primitive and consisted of a simple manager object that was governed by a synchronous lock (a synchronization function built into Java). As this was a prototype, the proxy was excessively simple in that it simply relayed data in both directions through the use of two threads. To improve performance of the proxy transport mode, much more intelligence would have to be built into the two threads (in addition to a communication conduit between the relay threads).

The improvements in the proxy threads would be to allow them to more efficiently relay data between the user and the remote server. Currently the relay threads only read and write bytes and pay no attention to the contents of the communication being passed

between the source and the destination. The sending of bytes by the relay thread causes the communication system to create many unnecessary packets (often a packet will be built around a single byte of transmitted data). The primary improvement would be to have the relay threads be capable of understanding the communications going on in the system. This would allow them to receive a full packet of information and relay it as a block instead of relaying it as discrete bytes. A communication conduit between the two threads would need to be established to allow the reply thread to react appropriately to the data being sent back to the user form the server. The outbound (user to server) thread would recognize the request being made, buffer and send the data and alert the reply thread that information is going to be coming back on the inbound link. The inbound thread could use this information to appropriately relay the information to the client in packet form.

6.1.3. AS Server

The AS server was created to allow the spawning of local applications. It was not intended to allow proxy (remote) service or allow the generic addition of applications to the user's application list. In this capacity (limited) it was successful in supplying the user(s) with applications from a local host.

This server was not intended to ever work in proxy for the user, and as such it would require significant modification. The multi-threaded operation of the server was tested and worked adequately.

6.2. Performance

As this system was not intended to do anything more than demonstrate the viability of the LID environment, the performance metrics of the servers was not at issue. They

performed their tasks without significant delay. If optimization were required, the UNS Server and the UNS API would have to be improved in terms of the packeting format and the general format of the data being passed. The same is true of the AS Server and the AS API.

In future work, the caching of requests and transmission of information through LAN and WAN networks will require significant study in order to optimize efficiency of the system. At such a time the optimized performance of the servers (UNS and AS) will be essential to the success of the system and user acceptability.

At present, the system performs with very little delay between request and the fulfillment of the request. In particular the system was tested with a slower modem (14.4 KBPS) as the primary communication link between the Desktop and the servers. In this mode of operation the system showed good response time and required at most no more that one or two seconds to respond to request. This is true even of the AS server which required the transmission of the largest volume of information in this projects (excluding the File System Service). At this time, the performance of the system was adequate to show the viability of the system.

Chapter 7

7. CONCLUSION AND FUTURE DIRECTIONS

7.1. General Conclusion

This thesis shows the viability of the Location Independent Desktop. It is simply created and is easily portable to different platforms. However, the performance of Java (particularly in the area of the AWT) leaves the question of Java's ability to support such a system open to discussion. In conclusion, the system performed well and could be a viable solution for actual system implementation.

7.2. Future Directions

The future enhancements are provided for those interested in future experimentation with this system. The basic enhancements are broken down into two categories: Basic System and System Augmentation. Basic System enhancements are the changes that are necessary for the system to move from a test system to a fully functional system. System Augmentation enhancements are expansions of the basic desktop premise.

7.2.1. Basic System Enhancements

7.2.1.1. Keys

Key System - The key system overall needs to be examined (now that a trial version has been built) for improvements. Specifically the areas of security, information containment and expansion would need to be explored. A more efficient or flexible system could be built out of the service/channel key system so that it would be capable of supporting a much larger variety of services and functions. Also, using the bytefield in

the keys to support encryption and other security measures would be a useful area of investigation.

Service Keys - Service keys could be improved to allow for some keys to act as structural elements in a hierarchical view of keys. This would be useful in display and management of very large lists of keys (probably more applicable to Applications than File systems). Perhaps instead the Hierarchy Manager could be improved to handle it instead of creating structural keys. Also, the addition of more fields to the service key to make it more flexible for use in the FSS system and in the desktop could be quite useful.

7.2.1.2. Desktop

Desktop System - The desktop needs to be taken from the basic application that it now is, to a full desktop operating system. This would require a much lower level implementation of the Java virtual machine and the establishment of a Java OS (for which the desktop would be the front end). This could optimize the performance of the system and provide a much more accurate picture of the system performance.

Interface Development - The current interface is quite rudimentary. The selection of applications and other listing operations need to be carried out in a much more structured (hierarchical) manner. The general flow of windows could be set up in a more intuitive format.

Process Management - The management of child process of the desktop is quite rudimentary. An investigation into mechanisms and flow control for the child process could make the system much faster and more controlled. In addition the child processes need to be tied more closely to user status (login/logout) and desktop operation.

7.2.1.3. UNS Server

Database Management The current UNS Server uses a very primitive information management system. A new system that could manage information not only for users but could be flexible in supporting multiple desktop types for the users would be required for significant improvement to be made in the development of multiple desktop types for this system.

UNS API - The information transmission and receipt mechanisms need to be optimized to improve performance.

Proxy Operation The current proxy mechanism is too simple and wastes significant transmission bandwidth by creating unnecessary packets due to the byte relay configuration under which it now operates. The proxy relay clients also could be expanded to handle error conditions in replies from the remote server (due to the unreliable nature of WANs).

7.2.1.4. AS Server

AS Server The Application server problem needs to be analyzed and rebuilt from scratch. The current mechanism (while proving simple and efficient transport) needs to be rebuilt with verification of users and proxy mechanisms as a primary concern. Caching needs to be addressed for commonly used applications.

7.2.1.5. Communications

Unreliable Communication The area of unreliable communications needs to be addressed. The creation of timeoutable ports, possibly moving to a packet format (in contrast to the current stream format) to facilitate error detection and recovery and system handling of erroneous communication. The current prototype relies on

completely reliable communication, this is a fallacy (especially in view of the fact that WAN communication is essential for system functionality).

Secure Communication - Current communication is completely open in that all information is transmitted in its original form. A secure, encrypted transmission would be required for significant system growth and acceptance. A possible point of entry could be a layer under the various communication APIs and on top of the standard port transmission layer.

7.2.2. System Augmentation Enhancements

7.2.2.1. Services

Stream Services - Streamed multimedia services are becoming quite popular [Abramowski 96] on the internet. They include such services as audio (both broadcast and point to point) and on demand video (both broadcast and point to point).

Printing - Creation of a server to handle printing and allow it to operate in the same manner as the current UNS and AS operate (i.e. with service keys) to allow users the ability to print anywhere.

Portmapper - Creation of a Portmapper-like solution for the desktop to find the local servers (reducing the need for any significant desktop specific configuration).

7.2.2.2. Desktop Distribution

Processor Distribution - Creation of a more distributed desktop environment to handle more processor intensive jobs (such as scientific applications), thus freeing the local processor to concentrate on the issues of user interaction and display.

Memory Distribution - Allow memory resources to be obtained from remote servers. This includes services such as paging across the network.

Bibliography

[Abramowski 96] Abramowski, S.; Klabunde, K.; Konrads, U.; Neunast, K.; Tjabben, H. *The Multimedia Reference Model: a Framework Facilitating the Creation of Multi-User, Multimedia Applications*, In Intelligent Networks and New Technologies. Proceedings of the IFIP TC6 Conference on Intelligent Networks and New Technologies, 1996, vi+307,265-280. Chapman & Hall. London, UK

[Bestavros 95] Bestavros, A.; *Demand-based Document Dissemination to Reduce Traffic and Balance Load in Distributed Information Systems*, In Proceedings. Seventh IEEE Symposium on Parallel and Distributed Processing (Cat. No.95TB8131), p. xvii+724, 338-45 IEEE Computing Soc. Press; Los Alamitos, CA, USA

[Dave 92] Dave, A.; Sefika, M.; Campbell, R.H.; *Proxies, Application Interfaces, and Distributed Systems*, In Proceedings of the Second International Workshop on Object Orientation in Operating systems (Cat. No.92TH0477-0), p. xi+384, 212-20, IEEE Computing Soc. Press; Los Alamitos, CA, USA

[Gellman 96] Gellman, R.; *Disintermediation and the Internet*, In Government Information Quarterly, Vol 14, Number 1, p. 1-8 JAI Press, Inc.

[Hamilton 96] Hamilton, M.; *Java and the Shift to Net Centric Computing*, In Computer, vol. 29, no. 8 31-39. IEEE Computing Society, August 1996

[Hardjono 93] Hardjono, T., Ohta, T.; *Fixed-Path Proxy-Logins with Assured Termination*, In Proceedings ICCI '93. Fifth International Conference on and Information (Cat. No.93TH0563-7), p. xvi+587, 111-15, IEEE Computing Soc. Press; Los Alamitos, CA, USA

[Hokimoto 96] Hokimoto, A., Kurihara, K., Nakajima, T.; *An Approach for Constructing Mobile Applications Using Service Proxies*, In Proceedings of the 16th International Conference on Computing

Systems (Cat. No.96CB35954), p. xviii+772, 726-33. IEEE Computing Soc. Press; Los Alamitos, CA, USA

[Jeffrey 96] Jeffrey, C., Samir, D.; *Proxy Sharing Proxy Servers*, In Proceedings. The First Annual Conference on Emerging and Applications in Communications (Cat. No.96TB100035), p. ix+205, 116-19. IEEE Computing Soc. Press; Los Alamitos, CA, USA

[Luotonen 94] Luotonen, A., Altis, K.; *World Wide Web Proxies*, In Computer Networks and ISDN Systems, vol.27, no.2, p. 147-54, CERN, Geneva, Switzerland

[Metzler 96] Metzler, B.; Miloucheva, I. *Design and Implementation of Flexible User Protocol Interface*, In Journal of High Speed Networks, vol.5, no.2, 1996, 145-158. IOS Press. Netherlands.

[Miller 97] Miller, K.; A *Java Based Internet File System Service*, Rochester Institute of Technology, Department of Computer Engineering, May, 1997

[Oguchi 96] Oguchi, M.; Ono, K.; *A Study of Caching Proxy Mechanisms Realized on Wide Area Networks*, In Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (Cat. No.TB100069), p. xviii+642, 443-9, IEEE Computing Soc. Press; Los Alamitos, CA, USA

[Shapiro 86] Shapiro, M.; *Structure and Encapsulation in Distributed Systems: The Proxy Principle*, In 6th International Conference on Distributed Computing Proceedings (Cat. No. 86CH2293-9), p. x+630, 198-204, IEEE Computing Soc. Press; Washington, DC, USA

[Tomes 96] Tomes, C. *Desktop Dreaming*, In Network World, vol 4, no. 9, October 1996, 26-28. IDG Communications, Australia

[Varadharajan 91] Varadharajan, V., Allen, P., Black, S.; *An Analysis of the Proxy Problem in Distributed Systems*, In Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy (Cat. No.91CH2986-8),. ix+389, 255-75, IEEE Computing Soc. Press; Los Alamitos, CA, USA

[Yourdon 96] Yourdon, E. *Java, the Web, and Software Development*, In Computer, vol. 29, no. 8 25-30. IEEE Computing Society, August 1996

[Youxun 96] Lei Youxun; Shi Yi; Meng Luoming; Chen Junliang *The Desgn and Implementation of Service Independent Configuration of User Service Data in an Intelligent Network*, In International Conference on Communication technology Proceedings. ICCT May 1996, 404-407. IEEE, New York, NY USA

APPENDIX A: INSTALLATION GUIDE

Desktop:

File Organization

The desktop uses many packages. The files for those packages must reside in subdirectories of the main working directory. They are case sensitive so care must be taken to name them properly. The only two files than need to reside in the main directory are:

Desktop.class

DTServer.txt

The desktop requires the following files in the listed sub directories:

Desktop
- MainDesktop.class
- LoginWindow.class
- User_Actions.class
- System_Actions.class
- Desktop_Manager.class
- Login_Manager.class
- Application_Manager.class
- Application_Display.class
- Application_Modify.class
- Service_Key_Constructor.class
- User_Config_Manager.class
- Config_Window.class
- File_System_Manager.class
- File_System_Display.class
- File_System_Modify.class
- FS_Service_Key_Constructor.class
- Execution_Manager.class

EXE_Relay_Thread.class

Keys
Channel_Key.class
Service_Key.class

UNSUtils
Heirarchy_Manager.class
UNS_API.class
Config_Manager.class

DesktopUtils
EXE_API.class

ASUtils
AS_API.class

Configuration

To configure the desktop only one file needs to be modified: DTServers.txt. This file contains the names of the local servers only desktop (since remote server are located via the Service Keys) that will service the. The file is text based and human readable. Its format is quite simple. The first line is the UNS Server. The second line is the AS Server. The third line is the FSS Server.

Example:

UNSServe.name.mane.edu
ASServe.name.mane.edu
FSServe.name.mane.edu

Starting the Desktop

Starting the desktop is quite simple. Simply make the working directory the main directory that has Desktop.class in it as well as DTServers.txt. Make sure that your environment has a path set up to your Java interpreter. Simply type the following to start the Desktop:

java Desktop

note: Remember the Java interpreter is case sensitive, so do not forget to capitalize Desktop.

UNS Server

File Organization

The UNS Server uses many packages. The files for those packages must reside in subdirectories of the main working directory. They are case sensitive so care must be taken to name them properly. The only file that needs to reside in the main directory is:

UNSServer.class

The UNS Server requires the following files in the listed sub directories:

UNSServer
- Mediator.class
- UNSClient.class
- BootStrap.class
- User_Record.class
- User_Database.class
- UNSCommManager.class
- Relay_Thread.class

Keys
- Channel_Key.class
- Service_Key.class

UNSUtils
- Heirarchy_Manager.class
- UNS_API.class
- Config_Manager.class
- User_Info_Manager.class

Configuration

The UNS Server has two external configuration files. They are UNSUser.dat and FSSHost.dat. UNSUsers.dat contains the names and passwords for each user for this UNS Server. The FSSHosts.dat contains the names of the FSS hosts and the root user directory for each host. The formats are quite simple.

UNSUsers.dat has a simple format. The user names and passwords must be in order and each on a separate line. So the basic file would have a list of pairs of strings. For example:

```
Sam_Snead
Quark
John_Doe
Foobar
Jane_Smith
Passwd
```

So to add a user just append the file with a new pair (user name and password) that indicates the user's preferences.

The FSSHosts.dat is also quite simple. It is arranged the similarly to the UNSUsers.dat file that of sets of strings (in this case three). The first string is the visible host name (for users to see), the second is the DNS host name, the third string is the root access path for that server. So the basic file would have a list of pairs of strings. For example:

```
New York
hpcims17.isc.rit.edu
/home/stu6/kam3011/FSSServer
San Francisco
hpcims18.isc.rit.edu
/home/stu6/kam3011/FSSServer
London
hpcims19.isc.rit.edu
```

/home/stu6/kam3011/FSSServer
Tokyo
hpcims20.isc.rit.edu
/home/stu6/kam3011/FSSServer

So, to add a FSS Host, just append the file with a new set (Host Name, DNS Host Name and access directory) that indicates the appropriate FSS information.

Starting the UNS Server

Starting the UNS Server is quite simple. Simply set the working directory to be the main directory that has UNSServer.class and the two configuration files (FSSHost.dat and UNSUser.dat) in it. Make sure that your environment has a path set up to your Java interpreter. Simply type the following to start the UNS Server:

java UNSServer

Note: Remember the Java interpreter is case sensitive, so do not forget to correctly capitalize UNSServer.

AS Server

File Organization

The AS Server uses many packages. The files for those packages must reside in subdirectories of the main working directory. They are case sensitive so care must be taken to name them properly. The only file that needs to reside in the main directory is:

ASServer.class

The AS Server requires the following files in the listed sub directories:

ASServer
    ASCommManager.class
    ASClient.class

Keys
    Channel_Key.class
    Service_Key.class

ASUtils
    AS_API.class

Configuration

The AS Server has no external configuration files. NO manual configuration is necessary.

Starting the AS Server

Starting the AS Server is quite simple. Simply make the working directory is the main directory that has ASServer.class in it. Make sure that your environment has a path set up to your Java interpreter. Simply type the following to start the AS Server:

java ASServer

Note: Remember the Java interpreter is case sensitive, so do not forget to correctly capitalize ASServer.

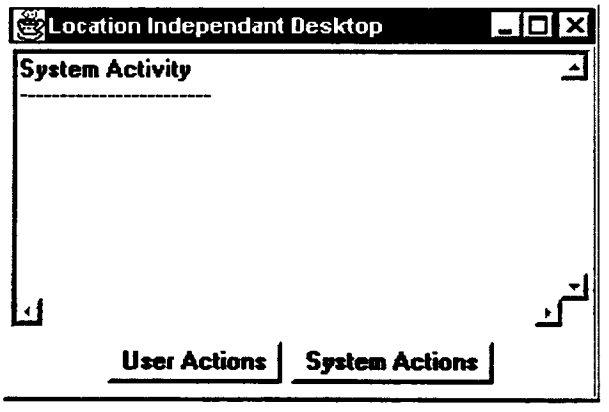
APPENDIX B: USER'S GUIDE

Introduction

This system is intended to provide an environment for exercising and manipulating the LID environment. It allows user access to all the major functions of the desktop environment through user selections. It does require a valid login to access all but the most basic system commands.

Note: This guide assumes that the desktop is started and operational. Refer to the installation guide for more details regarding the startup and maintenance of this system

The basic desktop (pictured below) has three main points of interest. They are the text display window and the two buttons. The text display window gives feedback to the user about the status of the system and the current action that is (or was) being performed.

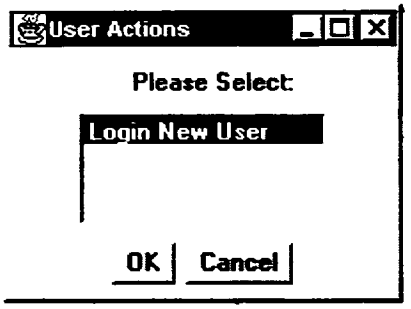


The two buttons ("User Actions" and "System Actions") provide access to the first layer of commands for the system. To activate a selection, simply click the mouse on the

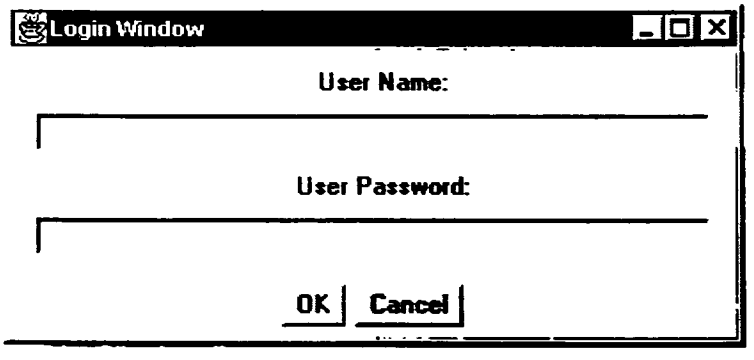
appropriate button. This window cannot be destroyed by simply clicking the 'X' on the frame because this window provides the root access to the system. The shutdown of the system will be discussed later.

Logging In and Out

To log in, simply click the User Actions button and the following window will appear (provided that you are not already logged in).



Simply press the "OK" button (the "Login" selection has already been highlighted for you), then the following window will appear. If you press cancel the window will be destroyed and no action will occur.



The you must fill in the user name and password. User names are not simple names, they also include a DNS address, for example:

username@machine.dom.aine

In the demonstration system there are three valid users: Sam_Snead, John_Doe, and Jane_Smith (more can be added later, see the Installation Guide). Their names should be concatenated with an @host.name for use in this system. User names without an '@' symbol will not be accepted by the system. The user name is the information to the left of the '@' symbol, it must be made up of simple printable characters excluding the '@' symbol. The host name must be a valid DNS address. A valid DNA address is usually made up of at least three parts: the machine name, the specific domain name and the general domain. The various parts are put together with dots. For an example:

Machine: tim

Specific Domain: rit

General Domain: edu

Result: tim.rit.edu

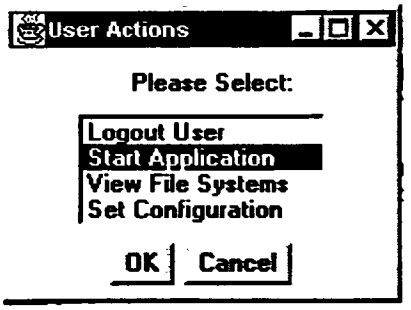
The passwords are simple, case sensitive strings. You may use any printable character in your password with the number of characters constrained to less that 256 characters.

The passwords for the demonstration users are:

Sam_Snead	Quark
John_Doe	Foobar
Jane_Smith	Passwd

Once the information has been entered, press the "OK" button. The main text window will inform you of the success or failure of this action.

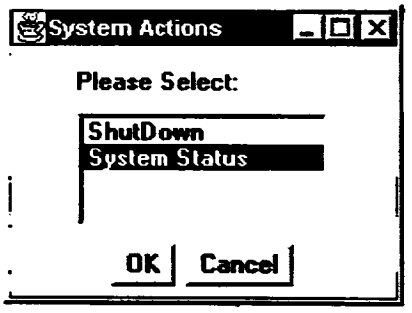
To log out, press the "User Actions" button (assuming that you are already logged in) and the following window will appear:



Select "Logout User" from the command list and press the "OK" button. The system will then log you out of the system. If you press the cancel button then you will not be logged out.

System Actions:

System actions are actions that modify the basic environment or display the status of the desktop. When the user is not logged in, pressing the "System Actions" button on the main screen makes the following window appear:



The user has two main choices: "ShutDown" and "System Status". These two choices will appear regardless of user login status.

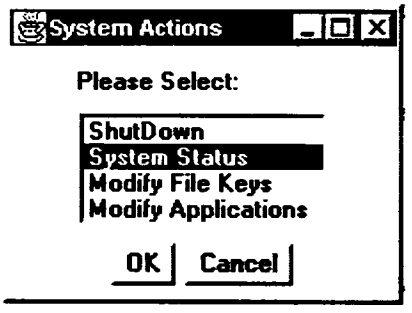
Shutting the system down

To shut the Desktop down select "ShutDown" from the menu and press "OK". This will (if anyone is logged in) log out the current user and cause the system to close all windows and cease operation.

System Status

To view the current system status select "System Status" from the list and press "OK". The system will then display (in the text area of the main window) the current user login status (and if the user is logged in, the user name) and a list of the three current local servers (UNS, FSS, and AS).

If the user is logged, in the following window will appear in response to a "System Actions" button press:



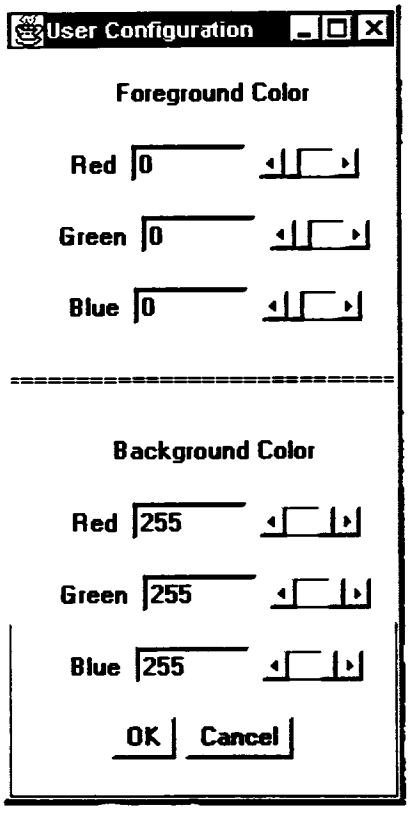
This allows the selection of two new choices: "Modify File Keys" (File System Section) and "Modify Applications" (Application Section) These choices will be discussed in the sections on Applications and File Systems.

User Actions

When a user is logged in, he or she has four choices to choose from in the User Actions window. They are: "Logout" (previously discussed), "Start Application" (Application Discussion), "View File Systems" (File System Section), and "Set Configuration".

Configuration

Choosing "Set Configuration" from the list on the User Actions window will result in the following window being displayed:



This window allows the viewing and changing of the foreground and background colors of the user desktop environment. (Note: currently this feature does nothing more than

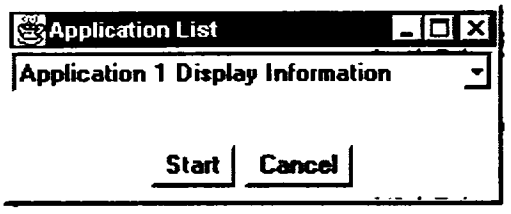
display some numbers due to problems with the Java windowing system (AWT) and its ability to update window colors.) In future revisions of the desktop, this feature should be implemented so that the changing of the values here will have visible effect on the system display. Changes to the values can be made by using the sliders on the right side of the numerical display. If "OK" is pressed, the system will accept the changes and write the changes all the way back to the UNS Server.

Applications

Applications are delivered to the desktop from the local AS Server. This server (at present) has two applications it can deliver. They are: Text Editor that can be accessed with the application name of "Application 1" or "Application 3" and the File Manager that can be accessed as "Application 2" (see [Miller 97] for a description).

Starting and Application

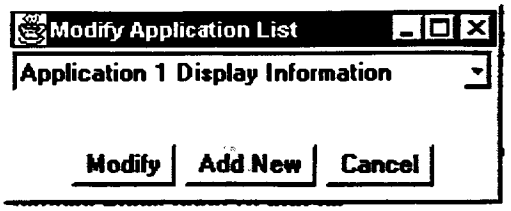
To start an application select "Start Application" from the User Actions window. The following window will then be displayed:



To select an application from the list, simply click the mouse on the down arrow at the right of the text box. Then, a list of the available applications will be displayed in a list box. Choose the application that is desired and press the "OK" button. The system will then attempt to obtain the application from the AS Server.

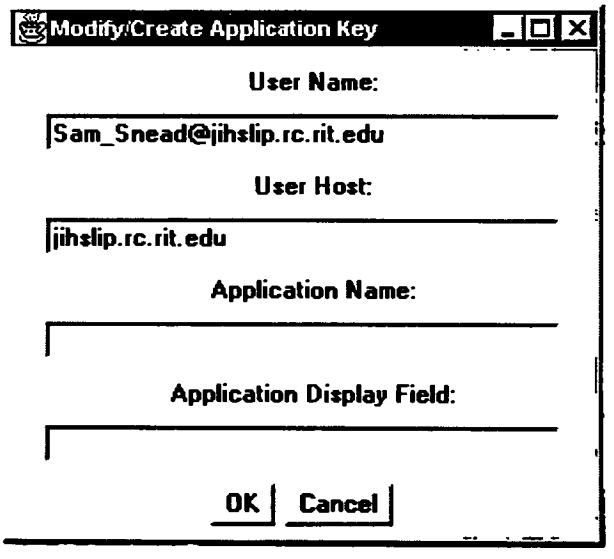
Addition and Modification of Applications

To modify the list of applications. Choose "Modify Applications" from the System Actions window. The following window will be displayed:



Simply select the Application to be modified from the list box by pressing the down arrow on the right of the text box and select the application to modify. Then press the "Modify" button. If you wish to create a new application then press the "Add New" button and a new application will be created and will be presented for modification.

Once "Modify" or "Add New" has been pressed the following window will be displayed:



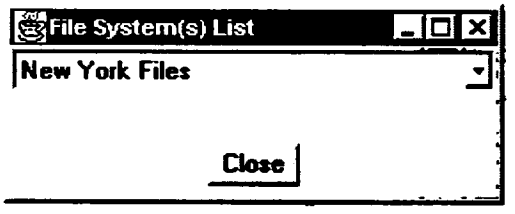
The User Name and User Host fields are fixed and cannot be modified. They are derived from the user user's login name. The Application Name field contains the name of the application that will be retrieved from the AS Server. The Application Display Field is what will be displayed to the user in the application start and application modify lists. Currently there are only three valid entries for Application Name; they are: "Application 1", "Application 2", and "Application 3". There is no restriction on what can be entered in the Application Display Field. Once the "OK" button has been pressed, the entries are saved, the local list is updated and the UNS Server is updated with the information. If "Cancel" is pressed then all changes are forgotten and no updates are made.

File Systems

File systems are accessed through applications (such as the file manager [Miller 97]). As such, they are not directly accessible to the user. Therefore the viewers only allow the display of the file system service keys and not the activation of the keys. Modification and addition of new keys is still provided.

Viewing File Systems

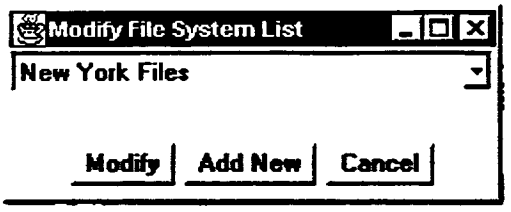
To view the file systems available to the user, select "View File Systems" from the User Actions window. The following window will be displayed:



The only actions that are possible are viewing the list and closing the window. To view the list press the arrow on the right of the text box, this will cause the list to appear. Then when the you are finished, the pressing of the "Close" button will close the window.

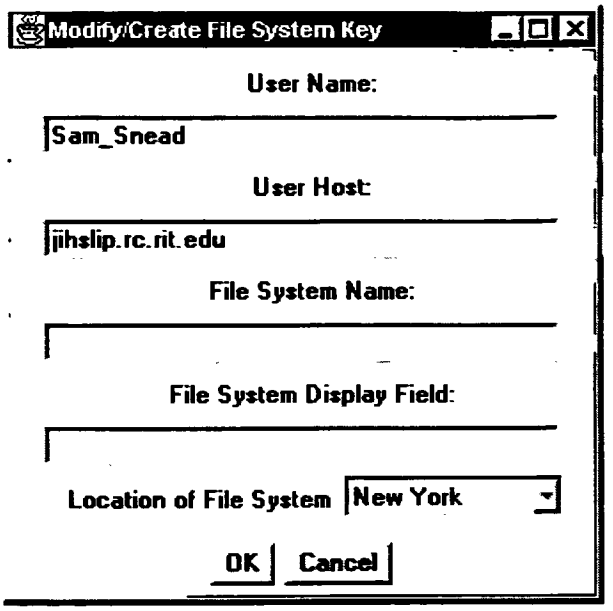
Addition and Modification of File Systems

To modify the list of applications. Choosing "Modify File Systems" from the System Actions window will cause the following window will be displayed:



Simply select the File System to be modified from the list box by pressing the down arrow on the right of the text box and select the File System to modify. Then press the "Modify" button. If you wish to create a new File System then press the "Add New" button and a new File System will be created and presented for modification.

Once "Modify" or "Add New" has been pressed the following window will be displayed:



The User Name and User Host fields are fixed and cannot be modified. They are derived from the user user's login name. The File System Name field contains the name of the File System that will be retrieved from the FSS Server. The File System Display Field is what will be displayed to the user in the file system view and file system modify lists. There is no restriction on entries in the File System Name field. There is no restriction on what can be entered in the Application Display field. The list selection for "Location of File System" allows the user to input the city that the file system will work for. Once the "OK" button has been pressed, the entries are saved, the local list is updated and the UNS Server is updated with the information. If "Cancel" is pressed then all changes are forgotten and no updates are made.

Text Editor Application:

The text editor application is intended to be a basic demonstration of the system's capability to retrieve and execute application from a remote server. As such its

APPENDIX C: PROGRAMMER'S GUIDE

Introduction

This document describes how to use the Application Programming Interface (API) objects provided with the Location Independent Desktop (LID), User Name Service Server (UNS), and the Application Service Server (AS Server) to access file systems on remote servers.

In addition to the APIs there are additional utility objects that are intended to allow conversation with the APIs and to allow simple passing of complex data structures. They are also the building blocks for secure user verification systems for services.

API interfaces are intended to simplify the job of communicating with remote servers and intercommunication between applications and the desktop environment. They are sometimes limited in action by the state of the API object. A detailed description of the APIs and their member methods follows:

APIs

UNS API

The UNS API is implemented as an object with a set of public member functions that carry out specific actions for the client. These actions are dependent on the state of the API object (in some cases) or may be used independent of the object status. The UNS API is intended to automate and simplify communication to the UNS Servers by

providing an interface to the server and thus insulating the user of the API from interaction specifics.

The general pattern of use is:

1. Instantiate UNS API
2. Log in new user
3. get configuration and service key lists
4. update service key lists and configuration
5. Log out user
6. repeat 2 - 5

Constructor

UNS_API (String Host):

Host indicates the DNS Name of the local UNS host. Can be set to null if the only use for the API is key verification. The local host MUST be specified for any valid actions to occur.

Login_User

boolean Login_User (String User_Name, String User_Password):

User_Name indicates the name of the user that is attempting to log in. The user name must contain an '@' symbol. All information to the left of the '@' is the user name and

all information to the right is used by the UNS to determine the primary host machine for the user and as such it has to contain a valid DNS address. The Password has to be a valid Java String that has a length of greater than 0. The UNS Server will examine the user name and password (if necessary it will contact a remote server to validate the user). The function will return a boolean that indicates the success or failure of the login attempt.

If the attempt succeeded, a channel between the UNS_API object and the UNS server has been established. This channel will be used by future requests for information. All other methods (except Verify_Key and the construictor) require that a successful login has taken place and the channel has been established for proper functionality.

Logout_User

boolean Logout_User ( ) :

Requires: There has been a successful login (without a corresponding logout).

Will attempt to terminate the connection between the UNS_API and the UNS Server that the user is logged into. Returns a true if the operation was successful.

Get_User_Heirarchy_Apps

Heirarchy_Manager Get_User_Heirarchy_Apps ( ) :

Requires: There has been a successful login (without a corresponding logout).

Attempts to contact the UNS Server to obtain the list of applications that the user can use. It returns the list in the form of a Heirarchy_Manager. This manager contains all the keys and allows access to them (for a more detailed description see the

Heirarchy_Manager description later in this section. Will return a null if any failure has occurred (if a failure has occurred the system probably has lost the connection with the UNS Server and a new login needs to take place).

Get_User_Heirarchy_File

Heirarchy_Manager Get_User_Heirarchy_File ( ) :

Requires: There has been a successful login (without a corresponding logout).

Attempts to contact the UNS Server to obtain the list of file systems that the user can use. It returns the list in the form of a Heirarchy_Manager. This manager contains all the keys and allows access to them (for a more detailed description see the Heirarchy_Manager description later in this section. Will return a null if any failure has occurred (if a failure has occurred the system probably has lost the connection with the UNS Server and a new login needs to take place).

Set_User_Heirarchy_Apps

boolean Set_User_Heirarchy_Apps (Heirarchy_Manager SaveManager) :

Requires: There has been a successful login (without a corresponding logout).

Attempts to contact the UNS Server to update the list of application that the user can access. The list must be in the form of a Heirarchy_Manager. It returns a true if the list has been updated successfully. Otherwise a failure occurred (if a failure has occurred the system probably has lost the connection with the UNS Server and a new login needs to take place).

Set_User_Heirarchy_File

boolean Set_User_Heirarchy_File (Heirarchy_Manager SaveManager) :

Requires: There has been a successful login (without a corresponding logout).

Attempts to contact the UNS Server to update the list of file systems that the user can access. The list must be in the form of a Heirarchy_Manager. It returns a true if the list has been updated successfully. Otherwise a failure occurred (if a failure has occurred the system probably has lost the connection with the UNS Server and a new login needs to take place).

Get_User_Configuration

Config_Manager Get_User_Configuration ( ) :

Requires: There has been a successful login (without a corresponding logout).

Attempts to contact the UNS Server to obtain the configuration for the user in the form of a Config_Manager. This manager contains all the information about the user's configuration and allows access to it (for a more detailed description see the Config_Manager description later in this section. Will return a null if any failure has occurred (if a failure has occurred the system probably has lost the connection with the UNS Server and a new login needs to take place).

Set_User_Configuration

boolean Set_User_Configuration (Config_Manager SaveConfig)

Requires: There has been a successful login (without a corresponding logout).

Attempts to contact the UNS Server to update user configuration. The configuration must be in the form of a Config_Manager. It returns a true if the list has been updated successfully. Otherwise a failure occurred (if a failure has occurred the system probably has lost the connection with the UNS Server and a new login needs to take place).

Verify_Service_Key

boolean Verify_Service_Key (Service_Key Test_Key)

This call does not require a valid login. If the user is logged in the connected UNS Server will be contacted to verify the key. If the user is not logged in the service key provided will be examined and the user's UNS host will be contacted to verify the key. The key provided must be valid and contain a valid user name and host.

For the key to be valid the following must be true:

1. The user must be logged in

2. The key provided is an EXACT copy of a key in the users application list or file system list.

If both of these conditions are true then the key is validated and true is returned. If not then false is returned.

EXE API

The EXE API is implemented as an object with a set of public member functions that carry out specific actions for the client. These actions are not dependent on any actions within the API. This API is intended to automate the communication between applications and the desktop environment. The communications between the desktop

and the application are carried out through System.in and System.out of the application. For this API to work properly, no reading or writing to the System ports (except error) is allowed.

The general pattern for use is:

1. Application Starts
2. Application instantiates EXE API
3. Get FSS Service keys from desktop
4. Get FSS Server and port
5. Use Write message to send any output to System.out
6. Application Closes (and destroys copy of API)

Constructor

EXE_API ( ) :

The constructor sets no state and performs no actions.

GetHeirarchy

Vector GetHeirarchy ( ) :

Returns a Vector whose members are Service Keys that are valid for use in FSS services. If the Vector is null, an error occurred and the application should probably exit. For a

further description of Vector, see the Java Package Description (not included in this document).

GetUserFSSHost

String GetUserFSSHost ( ) :

Returns a String that contains the host name for the local FSS Server. If a communication error occurred then null is returned and the application should probably exit.

GetUserFSSPort

int GetUserFSSPort ( ) :

Returns an Integer that contains the port number for the local FSS Server. If a communication error occurred then -1 is returned and the application should probably exit.

SendMessage

void SendMessage (String MessageOut) :

Allows applications to send text messages to the System.out of the desktop itself. The string must be valid. No status is returned.

AS API

The AS API is intended to allow users to interact with the AS Server to obtain runnable applications. The API is very simple and only contains three elements.

The general use pattern is:

1. Get or Construct a service key to use with AS API
2. Instantiate AS API with service key
3. Get Application
4. Close AS API
5. Save and start application

Constructor

AS_API (String Local_Host) :

Host indicates the DNS Name of the local UNS host. Can be set to null if the only use for the API is key verification. The local host MUST be specified for any valid actions to occur.

Start_New_Service

boolean Start_New_Service (Service_Key SKey) :

Attempts to begin a new service with the local AS Server. If the service key provided is validated with the UNS Server and contains valid information, the method return true and the channel to the server has been established.

Get_Application

byte [] Get_Application ( ) :

Requires: A successful Start_New_Service without a corresponding Close_Service.

Returns a buffer that contains the application and all the information needed to start it. If the buffer if null then an error occurred and the connection should propbably be closed and the application start attempt aborted.

The buffer format is as follows:

1. integer: The number of strings that indicate the process call name.

2. n Strings: The strings that indicate the process call name. (equal in number to the integer in 1)

3. integer: The number of file data blocks

4(a). String: the name of the file in this block

4(b). integer: the size of the data block for a file (in bytes)

4(c). n bytes: the for a file (size indicated in 4b)

example: the buffer for the Text Editor application contains:

```
2
java
TextEditor
1
TextEditor.class
n
n bytes
```

Close_Service

boolean Close_Service ( ) :

Attempts to close the open connection between the AS API and the AS Server. Returns true if successful or false if not. Either way the communication channel is closed and is no longer useable.

Utility Classes

Service_Key

The Service Key is the primary method of verifying user rights and privileges in the LID environment. It has six main fields: User Name, User Host, Host Port, Service Request, Additional Information, and the Byte Key. The User Name indicates the user who should be using this key, it should not include the User Host or an '@' symbol. The User Host indicates the UNS Host that originated the service key. The Host Port indicates the port number that the UNS host should be contacted at. The Service Request indicates the request that is being made with this key, the string may not be human readable, it should never be used for display purposes. The Additional Information is the information that is displayed to a user that is selecting this key out of a list. The Byte Key is a simple byte field of arbitrary dimensions that can be used by various services to store extra information and security information necessary to carry out or access the service.

Constructor (Null)

Service_Key ( ) :

Creates an empty service key. Access generally return null for all requests until information has been provided to fill in the key.

Constructor (Stream)

Service_Key (byte InStream []) :

Requires a valid byte stream to be passed in, the constructor will extract the information from the byte stream and store it appropriately.

Constructor (Copy)

Service_Key (Service_Key Original) :

Makes a duplicate of the original Service Key.

getUserName

String getUserName ( ) :

Returns the User Name stored in the Service Key or null if none set.

getUserHost

String getUserHost ( ) :

Returns the User Host stored in the Service Key or null if none set.

getServiceRequest

String getServiceRequest ( ) :

Returns the Service Request stored in the Service Key or null if none set.

getAdditionalInfo

String getAdditionalInfo ( ) :

Returns the Additional Information stored in the Service Key or null if none set.

getHostPort

int getHostPort ( ) :

Returns the Host Port stored in the Service Key or -1 if none set.

getKey

byte [] getKey ( ) :

Returns the Byte Key stored in the Service Key or null if none set.

getKeyStream

byte [] getKeyStream ( ) :

Returns the entire Service Key encoded into a byte stream (for transmission and packing purposes)..

setUserName

void setUserName (String sName) :

Allows the User Name to be set in the object. Requires that the String provided is valid.

setUserHost

void setUserHost (String sHost) :

Allows the User Host to be set in the object. Requires that the String provided is valid.

setServiceRequest

void setServiceRequest (String sRequest) :

Allows the Service Request to be set in the object. Requires that the String provided is valid.

setAdditionalInfo

void setAdditionalInfo (String sInfo) :

Allows the Additional Information to be set in the object. Requires that the String provided is valid.

setKey

void setKey (byte bKey[]) :

Allows the Byte Key to be set in the object. Requires that the byte array provided is valid.

setHostPort

void setHostPort (int Port) :

Allows the Host Port to be set in the object. Requires that the integer provided is valid.

CompareKeys (Stream)

boolean CompareKeys (byte BStream []) :

Returns true if the stream contains information that (when constituted into a Service Key) is equal to the current Service Key. Otherwise return false.

CompareKeys (Direct)

boolean CompareKeys (Service_Key OtherKey) :

Compares the submitted key with the current key information field by field. If all the information matches perfectly then true is returned, otherwise false is returned.

Channel_Key

The Channel Key is used to verify that communications between two hosts (that has already been established) is valid and that no foreign host is attempting use the communication conduit that is guaranteed by the channel key. It has two main pieces of information: User Name and the Byte Key. The User Name indicates for whom the channel was established for. The Byte Key contains an arbitrary byte field that can be used to validate the connection further (if necessary).

Constructor (Null)

Channel_Key ()

Constructs an empty Channel Key.

Constructor (Stream)

Channel_Key (byte InStream [])

Constructs a Channel Key from the information contained within the byte stream.

Constructor (Copy)

Channel_Key (Channel_Key Original)

Constructs a copy of the Channel Key provided in the argument.

getUserName

String getUserName ( ) :

Returns the User Name stored in the Channel Key or null if none set.

getKey

byte [] getKey ( ) :

Returns the Byte Key stored in the Channel Key or null if none set.

setUserName

void setUserName (String sName) :

Allows the User Name to be set in the object. Requires that the String provided is valid.

setKey

void setKey (byte bKey[]) :

Allows the Byte Key to be set in the object. Requires that the byte array provided is valid.

CompareKeys

boolean CompareKeys (Channel_Key OtherKey) :

Compares the submitted key with the current key information field by field. If all the information matches perfectly then true is returned, otherwise false is returned.

getKeyStream

byte [] getKeyStream ( ) :

Returns the entire Channel Key encoded into a byte stream (for transmission and packing purposes).

Hierarchy Manager

The Hierarchy Manager is an object that is intended to manage a list of Service Keys. It allows addition, deletion and access to the Service Keys contained with in it. It is the primary method that the UNS API uses to pass lists of keys through the inteface.

Constructor (Null)

Heirarchy_Manager ( ) :

Creates an empty Hierarchy manager.

Constructor (Byte Stream)

Heirarchy_Manager (byte InStream []) :

Creates a Hierarchy manager from the information contained within the data stream (byte buffer).

Constructor (Copy)

Public Heirarchy_Manager (Heirarchy_Manager Old_Heir) :

Creates a Hierarchy manager from the information contained within the other manager supplied

Constructor (Stream)

Heirarchy_Manager (DataInputStream InputStream) :

Creates a Hierarchy manager from the information contained within the data stream.

WriteHeirarchy

void WriteHeirarchy (DataOutputStream OutputStream) :

Packs all the information contained within the Hierarchy and passes it out on the Data Output Stream Provided.

getDataStream

byte [] getDataStream () :

Packs all the information contained within the Hierarchy and passes it out in the byte buffer it return. Returns null if any error occurs.

InsertAtIndex

Void InsertAtIndex (Service_Key NewKey, int Index) :

Attempts to replace the key at the given index with the key that has been provided.

AddKey

void AddKey (Service_Key NewKey) :

Add the provided key to the end of the current key list.

GetKeyIndex

Int GetKeyIndex (Service_Key NewKey) :

Determines if a key is in the list, if it is it returns the key's index. If it is not then it return.

DeleteKeyAtIndex

void DeleteKeyAtIndex (int Index) :

Delete the key at the index provided.

GetKeyAtIndex

Service_Key GetKeyAtIndex (int Index) :

Returns the key at the index give. Returns null for an invalid index.

GetTotalIndexs

int GetTotalIndexs ( ) :

Returns the total number of keys in the current list.

IsInList

boolean IsInList (Service_Key CheckKey) :

Return true if the supplied key is in the list, otherwise it return false.

Configuration Manager

Object that manages the configuration for a user. At this time it only contains two colors (foreground and background). Allows access and modification of the configuration.

Constructor (Null)

Config_Manager ( ) :

Builds a Configuration with the default colors: Foreground = Black and Background = White.

Constructor (Stream)

Config_Manager (DataInputStream InputStream) :

Builds a configuration manager with the information contained within a data stream.

Constructor (Byte Stream)

Config_Manager (byte InStream []) :

Builds a configuration manager with the information contained within the byte data stream.

Constructor (Copy)

Config_Manager (Config_Manager OldConfig) :

Builds a configuration manager with the information contained within the Configuration Manager supplied to it.

WriteStream

void WriteStream (DataOutputStream OutputStream) :

Write the information contained in the object to a data stream.

getDataStream

byte [] getDataStream ( ) :

Write the information contained in the object to a byte buffer

SetForegroundColor

void SetForegroundColor (Color NewFore) :

Set the foreground color contained in the Configuration Manager to the color specification argument.

SetBackgroundColor

void SetBackgroundColor (Color NewBack) :

Set the background color contained in the Configuration Manager to the color specification argument.

GetForegroundColor

Color GetForegroundColor ( ) :

Get the foreground color contained in the Configuration Manager.

GetBackgroundColor

Color GetBackgroundColor ( ) :

Get the background color contained in the Configuration Manager.

APPENDIX D: SOURCE CODE LISTING

Desktop Objects

File List:
Desktop_Manager.java
MainDesktop.java
LoginWindow.java
User_Actions.java
System_Actions.java
Desktop.java
Login_Manager.java
Application_Manager.java
Application_Display.java
Application_Modify.java
Service_Key_Constructor.java
User_Config_Manager.java
Config_Window.java
File_System_Manager.java
File_System_Display.java
File_System_Modify.java
FS_Service_Key_Constructor.java
Execution_Manager.java
EXE_Relay_Thread.java

Application_Manager

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class Application Manager
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Manages the list of application through a heirarchy manager (UNSUtils)
*   Controls display and modification of application list.
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ,
/**
*   System Imports
*/
import java.io.* ;
import java.net.* ;
/**
*   Custom Imports
*/
import Keys.* ,
import UNSServer.* ,
import UNSUtils.* ;
import Desktop.* ;

/**
*   Class Definition
*/
public
class Application_Manager
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // API -> UNS_API used to communicate with the UNS to
    //          obtain original keys and update keys
    private
    UNS_API API   null ,
    // KeyList -> List of application service keys to be
    //          used for this user
    private
    Heirarchy_Manager KeyList   null ,
```

```
// Parent -> Desktop Manager (Parent) for use in call backs
//          after system events
private
Desktop_Manager Parent   null ,


////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      inAPI -> the API that this object will use to
//               communicate with the UNS
//      Host -> Desktop Manager (Parent) for use in call backs
//               after system events
//
//  Get the user's application keys and set all the approporate
//  callback values.
public
Application_Manager ( UNS_API inAPI, Desktop_Manager Host )
{
    API   inAPI ;
    KeyList = API.Get_User_Heirarchy_Apps () ;
    Parent   Host ,
}

////////////////////////////////////////////
//  getUserInfo
//
//  Arguments:
//      none
//  Returns:
//      The user name and host from the parent
//
//  Used by service key modifiers/creators to get
//  user information about the system.
public String []
getUserInfo ()
{
    return Parent.GetUserInfo() ;
}

////////////////////////////////////////////
//  Display_Apps
//
//  Arguments:
//      none
//
//  Used by the Desktop Manager to cause the application
//  list to be displayed.
public void
Display_Apps ()
{
    // Create a list of Display Names for the display window
    int NumInList = KeyList.GetTotalIndexs () ,
    if ( NumInList > 0 )
    {
        String NameList []   new String [NumInList] ;
        for (int i   0 , i < NumInList , i++ )
        {
            Service_Key NewKey = KeyList.GetKeyAtIndex(i) ;
            NameList[i]   new String ( NewKey.getAdditionalInfo() ) ;
        }

        // Send the name list to the window manager to be displayed
        new Application_Display ( NameList, this ) ;
    }
}
```

```
////////////////////////////////////////////
//  Modify_Apps
//
//  Arguments:
//      none
//
//  Used to cause the application list to be displayed
//  modified (cause the modification window to appear).
public void
Modify_Apps ()
{
    int NumInList = KeyList.GetTotalIndexs () ,
    if ( NumInList > 0 )
    {
        String NameList []   new String [NumInList] ;
        for (int i   0 , i < NumInList , i++ )
        {
            Service_Key NewKey   KeyList.GetKeyAtIndex(i) ;
            NameList[i]   new String ( NewKey.getAdditionalInfo() ) ;
        }

        new Application_Modify ( NameList, this ) ;
    }
}

////////////////////////////////////////////
//  Display_Callback
//
//  Arguments:
//      Choice -> Integer that indicate whick key has been
//              chosen to start a new application
//.
//  Used to start a new application.
public void
Display_Callback ( int Choice )
{
    Parent.Start_Application( KeyList.GetKeyAtIndex(Choice) ) ,
}

////////////////////////////////////////////
//  Modify_Callback
//
//  Arguments:
//      Choice -> Integer that indicate which key has been
//              chosen to be modified ( -1 indicates that
//              a new key should be created).
//
//  Used to modufy the application key set.
public void
Modify_Callback ( int Choice )
{
    // Determine which key shoud be modified (if any)
    Service_Key SKey = null ,
    if ( Choice >= 0 )
        SKey   KeyList.GetKeyAtIndex(Choice) ,
    else
        SKey   null ,

    // Call the window manager that modifies/creates keys
    new Service_Key_Constructor ( SKey, this, Choice ) ,
}

////////////////////////////////////////////
//  Change_Callback
//
//  Arguments:
//      SKey -> Service key that is to inserted or added
//              to the list of application keys
```

```
    //      Index -> Integer that indicate which key has been
    //               chosen to be modified ( -1 indicates that
    //               a new key should be added to the list).
    //
    //  Used to modufy the application key set.
    public void
    Change_Callback ( Service_Key SKey, int Index )
    {
        // Decide where to insert the key and insert it
        if ( Index >= 0 )
        {
            KeyList.InsertAtIndex ( SKey, Index ) ,
        }
        else
        {
            KeyList.AddKey ( SKey ) ,
        }

        // Save the list back to the UNS
        API.Set_User_Heirarchy_Apps ( KeyList ) ;
    }

}
```

Application_Display

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class Application Display
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*       Displays a list of application (strings) and uses a callback routine
*       in its parent to return the selection (or lack there of) of the user
*       to the calling Class.
*
*       Display tailored to an application list
*
*       Extends Frame (for display attributes and functionality)
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Imports
*/
import java.awt.* ;
/**
*   Custom Imports
*/
import UNSUtils.* ;
import Desktop.* ,

/**
*   Class Definition
*/
public
class Application_Display extends Frame
{
    ////////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> OK Button on Screen
    private
    Button OK_but = null ,
    // Cancel_but -> Cancel Button on Screen
    private
    Button Cancel_but   null ;
    // List -> Application List
```

```
private
Choice Lister = null ;
// amParent -> Object to whom all call backs will be made
private
Application_Manager amParent   null ,

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InList -> Array of Strings that indicate applications
//      Parent -> Application Manager for callbacks
//
//  Builds Screen and sets up environment for callbacks.
//
public
Application_Display (String InList [], Application_Manager Parent )
{
    // Call Parents constructor
    super ( "Application List" ) ,

    // Set window layout method
    setLayout( new BorderLayout () ) ;

    // Create a new panel to go in window
    Panel top   new Panel () ,
    top.setLayout ( new BorderLayout () ) ;

    // Set the call back parent
    amParent = Parent ,

    // Crate a new choice box and fill it in
    // and add it to the top panel
    Lister   new Choice ( ) ;

    for ( int i = 0 , i < InList.length , i++ )
    {
        Lister.addItem ( InList[i] ) ,
    }

    Lister.select ( 0 ) ;
    top.add ( "Center", Lister ) ,

    // Create a new bottom panel and add the
    // buttons to it.
    Panel bottom = new Panel () ,

    OK_but   new Button("Start") ;
    Cancel_but = new Button("Cancel") ,

    bottom.add( OK_but ) ;
    bottom.add( Cancel_but ) ;

    // Add the panels to the window
    add("Center", top ) ;
    add("South", bottom ) ;
    resize(250, 100) ,

    // Set window attributes and show
    setResizable( false ) ;
    show () ,
}


////////////////////////////////////////////
//  handleEvent
//
//  Arguments
```

```
    //      evt -> System generated Event
    //
    //  Parent designated handler of all system events
    //  pertaining to this window.
    //
    public boolean
    handleEvent ( Event evt )
    {
        // Look for specific events
        if ( evt.target.equals ( OK_but ) )
        {
            // OK Button Clicked
            int ListChoice = Lister.getSelectedIndex () ,
            amParent.Display_Callback( ListChoice ) ;
            dispose () ,
        }
        else if ( evt.target.equals ( Cancel_but ))
        {
            // Cancel Button Clicked
            dispose () ;
        }
        else if ( evt.id == Event.WINDOW_DESTROY )
        {
            // X Frame button Clicked
            dispose () ,
        }
        return true ;
    }
}
```

Application_Modify

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class Application Modify
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows the user to select an application service key to be added or
*   modified.
*
*   Derived from Frame class
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop .
/**
*   System Import(s)
*/
import java.awt.* ,
/**
*   Custom Imports
*/
import UNSUtils.* ,
import Desktop.* ;

/**
*   Class Definition
*/
public
class Application_Modify extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> Modify Button on Screen
    private
    Button OK_but   null ;
    // Add_but -> Add button on screen
    private
    Button Add_but   null ;
    // Cancel_but -> Cancel button on screen
    private
    Button Cancel_but = null ,
    // List -> Application List
```

```
private
Choice Lister = null ;
// amParent -> Object to whom all call backs will be made
private
Application_Manager amParent   null ,

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InList -> Array of Strings that indicate applications
//      Parent -> Application Manager for callbacks
//
//  Builds Screen and sets up environment for callbacks.
//
public
Application_Modify (String InList [], Application_Manager Parent )
{
    // Construct the base window and set layout method
    super ( "Modify Application List" ) ,
    setLayout( new BorderLayout () ) ;

    // Create a panel to go in the window
    Panel top   new Panel () ;
    top.setLayout ( new BorderLayout () ) ,

    // Set the call back parent
    amParent   Parent ,

    // Crate a new choice box and fill it in
    // and add it to the top panel
    Lister   new Choice ( ) ;
    for ( int i   0 , i < InList.length , i++ )
    {
        Lister.addItem ( InList[i] ) ;
    }

    Lister.select ( 0 ) ;
    top.add ( "Center", Lister ) ,

    // Create a new panel and add the buttons to it
    Panel bottom = new Panel () ;

    OK_but   new Button("Modify") ;
    Add_but   new Button("Add New") ;
    Cancel_but   new Button("Cancel") ,

    bottom.add( OK_but ) ,
    bottom.add( Add_but ) ;
    bottom.add( Cancel_but ) ;

    // Add the panels to the main window
    add("Center", top ) ;
    add("South", bottom ) ,
    resize(250, 100) ,

    // Set window attributes and display window
    setResizable( false ) ,
    show () ,
}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
```

```
    //  pertaining to this window.
    //
    public boolean
    handleEvent ( Event evt )
    {
        // Determine if window event requires system action
        if ( evt.target.equals ( OK_but ) )
        {
            // Modify a selected entry
            int ListChoice   Lister.getSelectedIndex () ,
            amParent.Modify_Callback( ListChoice ) ;
            dispose () ,
        }
        else if ( evt.target.equals ( Add_but ))
        {
            // Add a new entry
            amParent.Modify_Callback( -1 ) ;
            dispose () ;
        }
        else if ( evt.target.equals ( Cancel_but ))
        {
            // Cancel button (just close window)
            dispose () ,
        }
        else if ( evt.id == Event.WINDOW_DESTROY )
        {
            // X Button on frame (just close window)
            dispose () ,
        }
        return true ,
    }
}
```

Config_Window

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class Config Window
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows the user to modify the current user configuration.
*
*   Derived from Frame class
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Import(s)
*/
import java.awt.* ,
/**
*   Custom Import(s)
*/
import Desktop.* ,

/**
*   Class Definition
*/
public
class Config_Window extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> OK Button on Screen
    private
    Button OK_but = null ;
    // Cancel_but -> Cancel button on screen
    private
    Button Cancel_but   null ,
    // Scroll1a -> Scroll Bar to control Red of foreground
    private
    Scrollbar Scroll1a = null ;
    // Scroll2a -> Scroll Bar to control Green of foreground
    private
    Scrollbar Scroll2a   null ,
```

```
// Scroll3a -> Scroll Bar to control Blue of foreground
private
Scrollbar Scroll3a   null ;
// Scroll1b -> Scroll Bar to control Red of background
private
Scrollbar Scroll1b = null ;
// Scroll2b -> Scroll Bar to control Green of background
private
Scrollbar Scroll2b = null ;
// Scroll3b -> Scroll Bar to control Blue of background
private
Scrollbar Scroll3b = null ;
// Number1a -> Text Field to display Red component of background
private
TextField Number1a   null ;
// Number2a -> Text Field to display Green component of background
private
TextField Number2a = null ;
// Number3a -> Text Field to display Blue component of background
private
TextField Number3a = null ;
// Number1b -> Text Field to display Red component of foreground
private
TextField Number1b   null ;
// Number2b -> Text Field to display Green component of foreground
private
TextField Number2b   null ;
// Number3b -> Text Field to display Blue component of foreground
private
TextField Number3b   null ;
// cmParent -> Where to update and get config from and to
private
User_Config_Manager cmParent = null ,

//////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      Parent -> Application Manager for callbacks
//      Front -> Color that is the current foreground
//      Back -> Color that is the current background
//
//  Builds Screen and sets up environment for callbacks.
//
public
Config_Window ( User_Config_Manager Parent, Color Front, Color Back )
{
    // Construct the base window and set layout method
    super ( "User Configuration" ) ,
    setLayout( new FlowLayout () ) ,

    // Create the full set of window panels
    Panel Top1 = new Panel () ,
    Panel A = new Panel () ,
    Panel B   new Panel () ;
    Panel C = new Panel () ;
    Panel Mid1 = new Panel () ,
    Panel Mid2 = new Panel () ,
    Panel D = new Panel () ;
    Panel E = new Panel () ;
    Panel F = new Panel () ;
    Panel bottom = new Panel () ,

    // Set the call back parent
    cmParent = Parent ,

    // Construct and set properties of text fields
    Number1a   new TextField ( ""+Front.getRed () , 5 ) ,
```

```
Number1a.setEditable( false );
Number2a = new TextField ( ""+Front.getGreen (), 5 ) ;
Number2a.setEditable( false );
Number3a = new TextField ( ""+Front.getBlue (), 5 ) ;
Number3a.setEditable( false );

// Construct and set properties of Scroll Bars
Scroll1a = new Scrollbar ( Scrollbar.HORIZONTAL, Front.getRed (), 32, 0, 255) ;
Scroll2a = new Scrollbar ( Scrollbar.HORIZONTAL, Front.getGreen (), 32, 0, 255) ,
Scroll3a   new Scrollbar ( Scrollbar.HORIZONTAL, Front.getBlue (), 32, 0, 255) ;

// Construct and set properties of text fields
Number1b   new TextField ( ""+Back.getRed (), 5 ) ;
Number1b.setEditable( false );
Number2b   new TextField ( ""+Back.getGreen (), 5 ) ;
Number2b.setEditable( false );
Number3b = new TextField ( ""+Back.getBlue (), 5 ) ,
Number3b.setEditable( false );

// Construct and set properties of Scroll Bars
Scroll1b   new Scrollbar ( Scrollbar.HORIZONTAL, Back.getRed (), 32, 0, 255) ;
Scroll2b   new Scrollbar ( Scrollbar.HORIZONTAL, Back.getGreen (), 32, 0, 255) ;
Scroll3b   new Scrollbar ( Scrollbar.HORIZONTAL, Back.getBlue (), 32, 0, 255) ,

// Add all the objects to the panels (+ Labels)
Top1.add ( new Label ("Foreground Color") ) ,
A.add ( new Label ("Red") ) ;
A.add ( Number1a ) ,
A.add ( Scroll1a ) ;
B.add ( new Label ("Green") ) ,
B.add ( Number2a ) ;
B.add ( Scroll2a ) ;
C.add ( new Label ("Blue") ) ,
C.add ( Number3a ) ;
C.add ( Scroll3a ) ;
Mid1.add ( new Label ("==============================") ) ,
Mid2.add ( new Label ("Background Color") ) ;
D.add ( new Label ("Red") ) ,
D.add ( Number1b ) ,
D.add ( Scroll1b ) ;
E.add ( new Label ("Green") ) ;
E.add ( Number2b ) ,
E.add ( Scroll2b ) ;
F.add ( new Label ("Blue") ) ;
F.add ( Number3b ) ,
F.add ( Scroll3b ) ,

// Create buttons and add them to the bottom panel
OK_but = new Button("OK") ,
Cancel_but = new Button("Cancel") ,
bottom.add( OK_but ) ;
bottom.add( Cancel_but ) ,

// Add all the panels to the window
add ( Top1 ) ,
add ( A ) ;
add ( B ) ;
add ( C ) ,
add ( Mid1 ) ;
add ( Mid2 ) ,
add ( D ) ;
add ( E ) ,
add ( F ) ,
add ( bottom ) ,

// Set window attributes and display
resize(200, 400) ;
setResizable( false ) ,
```

```
    show () ;
}

//////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean
handleEvent ( Event evt )
{
    if ( evt.target.equals ( Scroll1a ) )
    {
        // Scroll bar change
        Number1a.setText ( ""+Scroll1a.getValue () ) ;
    }
    else if ( evt.target.equals ( Scroll2a ) )
    {
        // Scroll bar change
        Number2a.setText ( ""+Scroll2a.getValue () ) ;
    }
    else if ( evt.target.equals ( Scroll3a ) )
    {
        // Scroll bar change
        Number3a.setText ( ""+Scroll3a.getValue () ) ,
    }
    else if ( evt.target.equals ( Scroll1b ) )
    {
        // Scroll bar change
        Number1b.setText ( ""+Scroll1b.getValue () ) ;
    }
    else if ( evt.target.equals ( Scroll2b ) )
    {
        // Scroll bar change
        Number2b.setText ( ""+Scroll2b.getValue () ) ,
    }
    else if ( evt.target.equals ( Scroll3b ) )
    {
        // Scroll bar change
        Number3b.setText ( ""+Scroll3b.getValue () ) ;
    }
    else if ( evt.target.equals ( OK_but ) )
    {
        // OK Button
        int ColorList []   new int [6] ,
        ColorList[0]   Scroll1a.getValue () ,
        ColorList[1] = Scroll2a.getValue () ;
        ColorList[2] = Scroll3a.getValue () ,
        ColorList[3]   Scroll1b.getValue () ,
        ColorList[4] = Scroll2b.getValue () ,
        ColorList[5]   Scroll3b.getValue () ;
        cmParent.UpdateColorsCallback ( ColorList ) ;
        dispose () ;
    }
    else if ( evt.target.equals ( Cancel_but ))
    {
        // Cancel Button
        dispose () ;
    }
    else if ( evt.id == Event.WINDOW_DESTROY )
    {
        // X button on frame
        dispose () ;
    }
```

```
            return true ,
        }
    }
```

Desktop

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Desktop (& Main)
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   3/24/97     1.0         Creation
*
*   Used to spawn a new instance of the L.I.D desktop.
*   Contains Main routine.
*/

/**
*   Custom Imports
*/
import Desktop.* ,

/**
*   Class Definition
*/
public
class Desktop
{
    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
    //      none
    //
    //  Builds a new Desktop_Manager to start the
    //  Desktop Environment
    //
    public Desktop ()
    {
        new Desktop_Manager () ,
    }

    ////////////////////////////////////////////
    //  Main
    //
    //  Arguments:
    //      args [] -> array of command line arguments
    //                 (ignored)
    //
    public static void main ( String args [] )
    {
        new Desktop ( ) ,
    }
```

```
}
```

Desktop_Manager

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*   Desktop Manager
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   3/24/97     1.0         Creation
*
*   Acts as central object in desktop management
*   system.  Most callbacks arrive here and are
*   processed.
*
*   Keeps limited track of state of environment.
*
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Imports
*/
import java.io.* ,
import java.awt.* ,
import java.net.* ;
/**
*   Custom Imports
*/
import Keys.* ;
import UNSServer.* ;
import UNSUtils.* ,
import Desktop.* ,

/**
*   Class Definition
*/
public
class Desktop_Manager
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // API -> API Interface to the UNS for the current user
    private
    UNS_API API = null ;
    // gDesktop -> Graphical desktop interface for the system
```

```
private
MainDesktop gDesktop = null ;
// LogManager -> Maintains login and logout
private
Login_Manager LogManager   null ;
// ApManager -> Application service key list manager
private
Application_Manager ApManager = null ;
// UCManager -> User Configuration Manager
private
User_Config_Manager UCManager = null ;
// FSManager -> File System service key list manager
private
File_System_Manager FSManager   null ;
// Current Logged in user's name
private
String User_Name   null ;
// Name of Local UNS Server
private
String LocalUNS   null ;
// Name of Local AS Server
public
String LocalAS = null ;
// Name of Local FSS Server
private
String LocalFSS   null ;
// File name of list of server names
private
String Server_ID_File   "DTServers.txt" ;
// Status of system (is a user logged in?)
private
boolean LoggedIn   false ;


////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      none
//
//  Builds and intializes windowing system,
//  user management system and callback system.
//
public
Desktop_Manager ( )
{
    try
    {
        // Open server file and load servers
        FileInputStream ServerFile   new FileInputStream ( Server_ID_File ) ;
        DataInputStream DataIn = new DataInputStream ( ServerFile ) ;
        LocalUNS = DataIn.readLine () ;
        LocalAS = DataIn.readLine () ;
        LocalFSS = DataIn.readLine () ,

        // Close file and allow to be garbage collected
        ServerFile.close () ,
        ServerFile = null ,
        DataIn = null ;

        // Create main system objects:
        //  Graphical Desktop
        gDesktop = new MainDesktop ( this ) ,
        //  UNS API
        API   new UNS_API ( LocalUNS ) ;
        //  Login/out manager
        LogManager   new Login_Manager ( API, this ) ;
```

```
        // Echo out Server names
        System.out.println ( "System Servers" ) ;
        System.out.println ( "-------------------------------------" ) ,
        System.out.println ( "Local UNS: "+LocalUNS ) ,
        System.out.println ( "Local AS: "+LocalAS ) ;
        System.out.println ( "Local FSS: "+LocalFSS ) ;
        System.out.println ( "-------------------------------------" ) ;

        // System now becomes event dependant
    }
    catch ( FileNotFoundException e )
    {
        System.exit(1) ,
        // Error loading server names, dosktop will not
        //  function without them so shut down.
    }
    catch ( IOException e )
    {
        System.exit(1) ,
        // Error loading server names, dosktop will not
        //  function without them so shut down.
    }
}

//////////////////////////////////////////
//  Action_Center
//
//  Arguments:
//      Action -> What has the user instructed
//              the system to do
//
//  Attempts to take appropriate action in response
//  to user commands.  May use client objects to
//  carry out commands.
//
public void
Action_Center ( int Action )
{
    // Use action value to determine what action to take
    switch ( Action )
    {
        case LOGIN
            LogManager.Login_User () ,
            // Login new user
            break ;
        case LOGOUT
            LogManager.Logout_User () ,
            // Log current user out
            break ;
        case SHUTDOWN
        {
            if ( IsLoggedIn () )
                LogManager.Logout_User () ;
            System.exit ( 0 ) ,
            // Kill the desktop
            break ,
        }
        case SYSTEM_STAT :
            Print_System_Status () ;
            // Print the system status
            break ;
        case MODIFY_FILE :
            FSManager.Modify_File () ,
            // Modify file service keys
            break ;
        case MODIFY_APPS :
            ApManager.Modify_Apps () ,
            // Modify Application service keys
```

```
            break ;
        case LIST_APPS :
            ApManager.Display_Apps () ;
            // Show all the Application and
            // allow user to start one
            break ;
        case LIST_FILES
            FSManager.Display_File () ,
            // Show all the FSS service keys
            break ;
        case SET_CONFIG :
            UCManager.Update_Config () ;
            // Show and allow update of
            // user configuration
            break ,
        default
            // Invalid command
            break ,
    }
}

////////////////////////////////////////////
//  IsLoggedIn
//
//  Arguments:
//      none
//
//  Allow read/only access to user login status
//
public boolean
IsLoggedIn ( )
{
    return LoggedIn ,
}

////////////////////////////////////////////
//  Print_System_Status
//
//  Arguments:
//      none
//
//  Display current login status and server list
//
private void
Print_System_Status ()
{
    gDesktop.DisplayMessage( "--------------------" ) ,
    if ( IsLoggedIn () )
    {
        gDesktop.DisplayMessage( "User: "+ User_Name ) ,
    }
    else
    {
        gDesktop.DisplayMessage( "No one logged in." ) ,
    }
    gDesktop.DisplayMessage( "UNS Host: "+ LocalUNS ) ,
    gDesktop.DisplayMessage( "AS Host: "+ LocalAS ) ;
    gDesktop.DisplayMessage( "FSS Host: "+ LocalFSS ) ;
    gDesktop.DisplayMessage( "--------------------") ,
}

////////////////////////////////////////////
//  GetUserInfo
//
//  Arguments:
//      none
//  Return:
//      Array of strings that contain the user name
```

```
//      and the user's UNS server.
//          index 0 -> User Name
//          index 1 -> User UNS Host
//
//  provide current user name and user UNS host
//
public String []
GetUserInfo ()
{
    String Result []   new String [2] ;
    Result[0] = new String ( User_Name ) ;
    Result[1] = new String ( ExtractUserHost () ) ,
    return Result ;
}

////////////////////////////////////////////
//  ExtractUserHost
//
//  Arguments:
//      none
//  Return:
//      String that contains the user's UNS host
//      that has been extracted from his/her login
//      name.
//
//  provide current user UNS host
//
private String
ExtractUserHost ()
{
    int Position = User_Name.indexOf( '@' ) ,
    String Result   new String ( User_Name.substring(Position+1) ) ;
    return Result ,
}

////////////////////////////////////////////
//  Update_Colors
//
//  Arguments:
//      Front -> Color for the foreground of the system
//      Back -> Color for the background of the system
//
//  attempt to update the color scheme of the windows
//
public void
Update_Colors ( Color Front, Color Back )
{
    gDesktop.UpdateColors(Front, Back) ,
}

////////////////////////////////////////////
//  Login_Callback
//
//  Arguments:
//      sucess -> is the user logged in?
//      inUser_Name -> if so what is the user name
//
//  used by login manager to update the system
//  login status.
//
public void
Login_Callback ( boolean sucess, String inUser_Name )
{
    // Set logged in
    LoggedIn = sucess ,
    if ( LoggedIn )
    {
        // Successful
```

```
        User_Name = new String ( inUser_Name ) ;
        ApManager = new Application_Manager ( API, this ) ,
        FSManager   new File_System_Manager ( API, this ) ,
        UCManager   new User_Config_Manager ( API, this ) ,
        gDesktop.DisplayMessage( User_Name + " logged in." ) ,
    }
    else
    {
        // Failed
        gDesktop.DisplayMessage( "Login failed or was canceled." ) ;
    }
}

////////////////////////////////////////////
//  Logout_Callback
//
//  Arguments:
//      sucess -> is the user logged out?
//
//  used by login manager to update the system
//  login status.
//
public void
Logout_Callback ( boolean sucess )
{
    LoggedIn   !sucess ;
    if ( !LoggedIn )
    {
        // sucess so clear out old user's config
        gDesktop.DisplayMessage( User_Name + " logged out." ) ,
        User_Name = null ;
        ApManager   null ;
        FSManager   null ,
    }
    else
    {
        // canceled or failed
        gDesktop.DisplayMessage( "Logout failed or was canceled." ) ;
    }
}

////////////////////////////////////////////
//  Start_Application
//
//  Arguments:
//      SKey -> Application service key fro the
//          applicatin that is to be started
//
//  used by application manager to request new ap.
//  startup.
//
public void
Start_Application ( Service_Key SKey )
{
    Execution_Manager ClientThread   new Execution_Manager ( SKey, this ) ,
    Thread CThread   new Thread ( ClientThread ) ,
    CThread.start () ,
}

////////////////////////////////////////////
//  GetFileSystemKeys
//
//  Arguments:
//      none
//  Returns:
//      Heirarchy_Manager (that contains the user's)
//              File System Keys
//
```

```
//  used by execution manager to request the user's
//  set of FSS service keys for the its application
//  use.
//
synchronized
public Heirarchy_Manager
GetFileSystemKeys () throws IOException
{
    return FSManager.GetFileKeys() ;
}

////////////////////////////////////////////
//  GetFSSHost
//
//  Arguments:
//      none
//  Returns:
//      String -> Name of FSS local host
//
//  used by execution manager to request the
//  FSS local server
//
synchronized
public String
GetFSSHost ()
{
    return new String ( LocalFSS ) ;
}

////////////////////////////////////////////
//  GetFSSPort
//
//  Arguments:
//      none
//  Returns:
//      int -> Name of FSS local host port
//
//  used by execution manager to request the
//  FSS local server port
//
synchronized
public int
GetFSSPort ()
{
    return FSSHostPort ,
}

////////////////////////////////////////////////////
//Public Member Variables
//
// Local FSS Host Port
public static final
int FSSHostPort   3011 ,
//
//  Folowing used in Action_Center for requesting various
//  actions:
public static final
int LOGIN     0 ;
public static final
int LOGOUT     1 ,
public static final
int SHUTDOWN     2 ;
public static final
int SYSTEM_STAT   = 3 ,
public static final
int MODIFY_FILE   = 4 ;
public static final
int MODIFY_APPS   = 5 ,
```

```
    public static final
    int LIST_APPS    = 6 ,
    public static final
    int LIST_FILES      7 ;
    public static final
    int SET_CONFIG    = 8 ,
}
```

Execution_Manager

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Execution Manager Object
*
*   Jeff Harman
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Starts a new execuatable that is downloaded from the ASServer
*   Uses the service key that is proviced as the basis for the
*   Application Request.
*
*   Extends Thread
*
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ,
/**
*   System Imports
*/
import java.io.* ,
import java.net.* ;
/**
*   Custom Imports
*/
import Keys.* ;
import UNSServer.* ,
import UNSUtils.* ,
import Desktop.* ;
import ASUtils.* ;

/**
*   Class Definition
*/
public
class Execution_Manager extends Thread
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // Parent -> Object to make system request to
    private
    Desktop_Manager Parent = null ,
    // SKey -> Applications Service Key
    private
```

```
Service_Key SKey = null ;
// API -> Application Server API object
private
AS_API API   null ;
// Child -> Executed Child process
private
Process Child   null ,


////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      AppKey -> Application to be started servcie key
//      Host -> Man desktop (for system info requests)
//
//  Builds Execuatabe, executes it,
//  and sets up environment for callbacks.
//
public
Execution_Manager ( Service_Key AppKey, Desktop_Manager Host )
{
    API   new AS_API ( Host.LocalAS ) ;
    Parent   Host ,
    SKey   AppKey ,
}

////////////////////////////////////////////
//  run
//
//  Arguments
//      none
//
//  (1) Establish connection to AS
//  (2) Get the application
//  (3) Close the AS Connection
//  (4) Run the application
//
public void
run ()
{
    // (1)
    if ( API.Start_New_Service ( SKey ) )
    {
        // (2)
        byte FileBuffer []   API.Get_Application () ,

        if ( FileBuffer != null )
        {
            // (3)
            if ( API.Close_Service () )
            {
                // (4)
                RunApplication ( FileBuffer ) ,
            }
        }
    }
}

////////////////////////////////////////////
//  RunApplication
//
//  Arguments:
//      FileBuffer -> buffer from AS that contains
//          all info and files for execution
//
//  Sets up the execuatabe, runs it, and listens for
//      information requests.
```

```
//
private void
RunApplication ( byte FileBuffer [] )
{
    try
    {
        // System out setup
        System.out.println ( "Application Output" ) ;
        System.out.println ( "------------------" ) ,

        // Create Application on local disk
        String ExeArray []   Save_Applcation ( FileBuffer ) ;

        // Actually start the application
        Child   Runtime.getRuntime().exec ( ExeArray ) ;

        // Get and capture childs System IO streams
        InputStream rawInput   Child.getInputStream () ;
        OutputStream rawOutput   Child.getOutputStream () ,
        InputStream rawError   Child.getErrorStream () ,

        DataInputStream DataIn   new DataInputStream ( rawInput ) ;
        DataOutputStream DataOut   new DataOutputStream ( rawOutput ) ,
        DataInputStream ErrorIn   new DataInputStream ( rawError ) ;

        // Echo error out to main err out
        EXE_Relay_Thread Relay = new EXE_Relay_Thread ( ErrorIn ) ,
        Thread CThread   new Thread ( Relay ) ;
        CThread.start () ,
        CThread   null ;
        Relay   null ,

        // Listen for requests from child
        // and act upon them
        boolean LoopStill = true ,
        while ( LoopStill )
        {
            // Get next command
            int Command   DataIn.readInt ( ) ;

            // Decide which command it is
            switch ( Command )
            {
                // Text to system.out
                case TEXTOUTPUT
                    String Output   DataIn.readLine ( ) ;
                    if ( Output != null )
                    {
                        System.out.println ( Output ) ,
                    }
                    else
                    {
                        LoopStill   false ,
                    }
                    break ;
                case GETHEIRARCHY
                    // Request for file system keys
                    SendHeirarchy ( DataOut ) ,
                    DataOut.flush () ;
                    rawOutput.flush () ,
                    break ;
                case GETFSSHost :
                    // Request for local FSS Host
                    DataOut.writeUTF ( Parent.GetFSSHost () );
                    DataOut.flush () ;
                    rawOutput.flush () ;
                    break ;
                case GETFSSPort
```

```
                    // Request for local FSS Host Port
                    DataOut.writeInt ( Parent.GetFSSPort () );
                    DataOut.flush () ;
                    rawOutput.flush () ,
                    break ;
                default
                    // invalid command
                    break ;
            }
        }

        System.out.println ( "Application Closed" ) ,
    }
    catch ( IOException e )
    {
        // error reading port / system closing
        System.out.println ( "Application Closed" ) ;
    }
}

private void
SendHeirarchy ( DataOutputStream DataOut ) throws IOException
{
    Heirarchy_Manager HMan   Parent.GetFileSystemKeys() ,

    HMan.WriteHeirarchy ( DataOut ) ;

    DataOut.flush () ,
}

////////////////////////////////////////////
//  Save_Applcation
//
//  Arguments:
//      FileBuffer -> buffer from AS that contains
//          all info and files for execution
//
//  Parse buffer, save files and extract execuatable
//  strings.
//
private String []
Save_Applcation ( byte FileBuffer [] ) throws IOException
{
    // Create data input streams to ease data
    // extractions from buffer
    ByteArrayInputStream Byter   new ByteArrayInputStream (FileBuffer) ;
    DataInputStream DataIn = new DataInputStream ( Byter ) ,

    // Extract number of name elements in buffer
    // and extract them
    int NameElements = DataIn.readInt () ,

    String ExeStrings []   new String [ NameElements ] ,

    for ( int i = 0 , i < NameElements , i++ )
    {
        ExeStrings[i]   new String ( DataIn.readUTF () ) ,
    }

    // Determine the number of files in the data stream
    // and extract them to local disk
    int FilesInStream   DataIn.readInt () ;

    for ( int i = 0 , i < FilesInStream , i++ )
    {
        String FileName   new String ( DataIn.readUTF () ) ,
        int Length = DataIn.readInt () ;
        byte FBuff [] = new byte [ Length ] ;
```

```
            DataIn.read( FBuff, 0, Length ) ;

            FileOutputStream Filer = new FileOutputStream ( FileName ) ;
            Filer.write ( FBuff, 0, Length ) ;
            Filer.flush () ,
            Filer.close () ;
        }

        return ExeStrings ;
    }

    ////////////////////////////////////////////
    //  RunApplication
    //
    //  Arguments:
    //      none
    //
    // System Closing so kill all children
    //
    protected void
    finalize()
    {
        Child.destroy () ,
    }

    ////////////////////////////////////////////////////
    //Public Member Variables
    //
    //  Folowing used by child processes for requesting various
    //  actions:
    public final static
    int TEXTOUTPUT   0 ,
    public final static
    int GETHEIRARCHY   1 ;
    public final static
    int GETFSSHost = 2 ;
    public final static
    int GETFSSPort   3 ,
}
```

EXE_Relay_Thread

```
/*****************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*****************************************************************************/

/*****************************************************************************
*   Executable Error Out Relay Thread Object
*
*   Jeff Harman
*   Date        Revision    Comment
*   4/11/97     1.0         Creation
*
*   Relays applcation's system.err output to the err output of the
*   Desktop through the Thread mechanism of blocking on reads
*   when no data is available.
*
*   Extends Thread
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Imports
*/
import java.io.* ;

public
class EXE_Relay_Thread extends Thread
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // DataIn -> data stream from clients err out
    private
    DataInputStream DataIn = null ,
    // DataOut -> data stream to desktop err out
    private
    DataOutputStream DataOut = null ;

    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
    //      InInformation -> Data Stream that
    //                  comes fromt he client err out
    //
    //  Sets up data ports.
    //
    public
    EXE_Relay_Thread ( DataInputStream InInformation )
```

```
    {
        DataIn = InInformation ;
        DataOut = new DataOutputStream(System.err) ;
    }

    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
    //      none
    //
    //  Relays data to err out.  Relies on exception
    //  mechanism to end execution.
    //
    public
    void run ()
    {
        try
        {
            while ( true )
            {
                DataOut.writeByte ( DataIn.readByte () ) ,
            }
        }
        catch ( IOException f )
        {
            DataOut   null ;
            DataIn   null ,
        }
    }
}
```

File_System_Manager

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class Application Manager
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Manages the list of application through a heirarchy manager (UNSUtils)
*   Controls display and modification of application list.
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ,
/**
*   System Imports
*/
import java.io.* ;
import java.net.* ,
/**
*   Custom Imports
*/
import Keys.* ,
import UNSServer.* ,
import UNSUtils.* ;
import Desktop.* ,

/**
*   Class Definition
*/
public
class File_System_Manager
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // API -> UNS_API used to communicate with the UNS to
    //          obtain original keys and update keys
    private
    UNS_API API = null ;
    // KeyList -> List of application service keys to be
    //          used for this user
    private
    Heirarchy_Manager KeyList = null ;
    // Parent -> Desktop Manager (Parent) for use in call backs
```

```
//          after system events
private
Desktop_Manager Parent = null ;

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      inAPI -> the API that this object will use to
//               communicate with the UNS
//      Host -> Desktop Manager (Parent) for use in call backs
//               after system events
//
//  Get the user's file system keys and set all the approporate
//  callback values.
public
File_System_Manager ( UNS_API inAPI, Desktop_Manager Host )
{
    API = inAPI ;
    KeyList   API.Get_User_Heirarchy_File () ;
    Parent = Host ,
}

////////////////////////////////////////////
//  getUserInfo
//
//  Arguments:
//      none
//  Returns:
//      The user name and host from the parent
//      and Strip Host name from User_Name
//
//  Used by service key modifiers/creators to get
//  user information about the system.
public String []
getUserInfo ()
{
    String UserInfo []   Parent.GetUserInfo() ;
    int Position = UserInfo[0].indexOf( '@' ) ;
    UserInfo[0]   new String ( UserInfo[0].substring(0,Position) ) ,
    return UserInfo ;
}


////////////////////////////////////////////
//  Display_File
//
//  Arguments:
//      none
//
//  Used by the Desktop Manager to cause the file
//  list to be displayed.
public void
Display_File ()
{
    // Create a list of Display Names for the display window
    int NumInList = KeyList.GetTotalIndexs () ;
    if ( NumInList > 0 )
    {
        String NameList []   new String [NumInList] ,
        for (int i   0 , i < NumInList , i++ )
        {
            Service_Key NewKey   KeyList.GetKeyAtIndex(i) ;
            NameList[i]   new String ( NewKey.getAdditionalInfo() ) ,
        }

        // Send the name list to the window manager to be displayed
        new File_System_Display ( NameList, this ) ,
```

```
    }
}

////////////////////////////////////////////
//  Modify_File
//
//  Arguments:
//      none
//
//  Used to cause the application list to be displayed
//  modified (cause the modification window to appear).
public void
Modify_File ()
{
    int NumInList   KeyList.GetTotalIndexs () ;
    if ( NumInList > 0 )
    {
        String NameList [] = new String [NumInList] ;
        for (int i   0 ; i < NumInList , i++ )
        {
            Service_Key NewKey   KeyList.GetKeyAtIndex(i) ;
            NameList[i]   new String ( NewKey.getAdditionalInfo() ) ;
        }

        new File_System_Modify ( NameList, this ) ,
    }
}

////////////////////////////////////////////
//  Modify_Callback
//
//  Arguments:
//      Choice -> Integer that indicate which key has been
//              chosen to be modified ( -1 indicates that
//              a new key should be created).
//
//  Used to modify the file system key set.
public void
Modify_Callback ( int Choice )
{
    Service_Key SKey   null ;
    if ( Choice >= 0 )
        SKey   KeyList.GetKeyAtIndex(Choice) ;
    else
        SKey   null ;

    new FS_Service_Key_Constructor ( SKey, this, Choice ) ;
}

////////////////////////////////////////////
//  Change_Callback
//
//  Arguments:
//      SKey -> Service key that is to inserted or added
//              to the list of file system keys
//      Index -> Integer that indicate which key has been
//              chosen to be modified ( -1 indicates that
//              a new key should be added to the list).
//
//  Used to modufy the file system key set.
public void
Change_Callback ( Service_Key SKey, int Index )
{
    if ( Index >= 0 )
    {
        KeyList.InsertAtIndex ( SKey, Index ) ,
    }
    else
```

```
        {
            KeyList.AddKey ( SKey ) ,
        }

        API.Set_User_Heirarchy_File ( KeyList ) ,
    }

    ////////////////////////////////////////////
    //  Change_Callback
    //
    //  Arguments:
    //      none
    //
    //  Get a copy of the file system Heirarchy File Manager
    //
    public Heirarchy_Manager
    GetFileKeys () throws IOException
    {
        return new Heirarchy_Manager ( KeyList ) ,
    }
}
```

File_System_Display

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class File System Display
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*       Displays a list of file systems (strings) and uses a callback routine
*       in its parent to return the selection (or lack there of) of the user
*       to the calling Class.
*
*       Display tailored to an files system list
*
*       Extends Frame (for display attributes and functionality)
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ,
/**
*   System Imports
*/
import java.awt.* ,
/**
*   Custom Imports
*/
import UNSUtils.* ;
import Desktop.* ,

/**
*   Class Definition
*/
public
class File_System_Display extends Frame
{
    ////////////////////////////////////////////////////////
    //Private Member Variables
    //
    // Cancel_but -> Cancel Button on Screen
    private
    Button Cancel_but   null ;
    // List -> File system List
    private
    Choice Lister = null ;
    // amParent -> Object to whom all call backs will be made
```

```
private
File_System_Manager amParent = null ;

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InList -> Array of Strings that indicate file systems
//      Parent -> Application Manager for callbacks
//
//  Builds Screen and sets up environment for callbacks.
//
public
File_System_Display (String InList [], File_System_Manager Parent )
{
    // Create main window and set attributes
    super ( "File System(s) List" ) ,
    setLayout( new BorderLayout () ) ,

    // Create a new panel and its attributes
    Panel top   new Panel () ,
    top.setLayout ( new BorderLayout () ) ;

    // Save the callback object
    amParent = Parent ,

    // Create the display list and fill it in
    Lister   new Choice ( ) ;
    for ( int i   0 , i < InList.length , i++ )
    {
        Lister.addItem ( InList[i] ) ;
    }
    Lister.select ( 0 ) ,

    // Add the list to the top window
    top.add ( "Center", Lister ) ,

    // Create new bottom panel (for buttons)
    Panel bottom   new Panel () ,

    // Create button and add it
    Cancel_but   new Button("Close") ,
    bottom.add( Cancel_but ) ,

    // Add panels to window, size window and display it
    add("Center", top ) ;
    add("South", bottom ) ,
    resize(250, 100) ;
    setResizable( false ) ,
    show () ,
}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean
handleEvent ( Event evt )
{
    if ( evt.target.equals ( Cancel_but ))
    {
        dispose () ;
    }
```

```
        else if ( evt.id == Event.WINDOW_DESTROY )
        {
            dispose () ,
        }
        return true ;
    }
}
```

File_System_Modify

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITHESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class File System Modify
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows the user to select an file system service key to be added or
*   modified.
*
*   Derived from Frame class
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Import(s)
*/
import java.awt.* ;
/**
*   Custom Imports
*/
import UNSUtils.* ;
import Desktop.* ,

/**
*   Class Definition
*/
public
class File_System_Modify extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> Modify Button on Screen
    private
    Button OK_but = null ;
    // Add_but -> Add button on screen
    private
    Button Add_but = null ;
    // Cancel_but -> Cancel button on screen
    private
    Button Cancel_but   null ;
    // List -> File System List
```

```
private
Choice Lister = null ;
// amParent -> Object to whom all call backs will be made
private
File_System_Manager amParent   null ;

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InList -> Array of Strings that indicate file systems
//      Parent -> Application Manager for callbacks
//
//  Builds Screen and sets up environment for callbacks.
//
public
File_System_Modify (String InList [], File_System_Manager Parent )
{
    // Construct the base window and set layout method
    super ( "Modify File System List" ) ,
    setLayout( new BorderLayout () ) ;

    // Create a panel to go in the window
    Panel top   new Panel () ;
    top.setLayout ( new BorderLayout () ) ,

    // Set the call back parent
    amParent   Parent ,

    // Crate a new choice box and fill ıt in
    // and add it to the top panel
    Lister   new Choice ( ) ,

    for ( int i   0 ; i < InList.length , i++ )
    {
        Lister.addItem ( InList[i] ) ;
    }

    Lister.select ( 0 ) ;
    top.add ( "Center", Lister ) ,

    Panel bottom   new Panel () ,

    // Create a new panel and add the buttons to it
    OK_but   new Button("Modify") ;
    Add_but   new Button("Add New") ;
    Cancel_but   new Button("Cancel") ,

    bottom.add( OK_but ) ,
    bottom.add( Add_but ) ;
    bottom.add( Cancel_but ) ,

    // Add the panels to the main window
    add("Center", top ) ;
    add("South", bottom ) ,

    // Set window attributes and display window
    resize(250, 100) ;
    setResizable( false ) ,
    show () ,
}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
```

```
    //  Parent designated handler of all system events
    //  pertaining to this window.
    //
    public boolean handleEvent ( Event evt )
    {
        // Determine if window event requires system action
        if ( evt.target.equals ( OK_but ) )
        {
            // Modify a selected entry
            int ListChoice = Lister.getSelectedIndex () ,
            amParent.Modify_Callback( ListChoice ) ;
            dispose () ;
        }
        else if ( evt.target.equals ( Add_but ))
        {
            // Add a new entry
            amParent.Modify_Callback( -1 ) ,
            dispose () ;
        }
        else if ( evt.target.equals ( Cancel_but ))
        {
            // Cancel button (just close window)
            dispose () ,
        }
        else if ( evt.id == Event.WINDOW_DESTROY )
        {
            // X Button on frame (just close window)
            dispose () ,
        }
        return true ;
    }
}
```

FS_Service_Key_Constructor

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class FS_Service_Key_Constructor
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows the user to modify or create a FSS service key.
*
*   Derived from Frame class
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Import(s)
*/
import java.awt.* ;
import java.io.* ,
/**
*   Custom Imports
*/
import UNSUtils.* ,
import Desktop.* ,
import Keys.* ;

/**
*   Class Definition
*/
public
class FS_Service_Key_Constructor extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> Modify Button on Screen
    private
    Button OK_but   null ;
    // Cancel_but -> Cancel button on screen
    private
    Button Cancel_but = null ,
    // amParent -> Object to whom all call backs will be made
    private
    File_System_Manager amParent   null ;
```

```
// UserName -> Text field that shows the user name in the service key
private
TextField UserName   null ;
// UserHost -> Text field that shows the user's host name in the service key
private
TextField UserHost   null ;
// AppService -> Text field that shows file key name
private
TextField AppService = null ;
// AppService -> Text field that shows file key display information
private
TextField AppDisplay   null ;
// List -> List of FSS Hosts
private
Choice Lister   null ;
// ListLocation -> Location of FSS Service key that the user has
//      chosen to modify ( -1 if creating new )
private
int ListLocation = 0 ,

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InKey -> Service key of FSS service key to be modified
//                  ( null if new key creation )
//      Parent -> Application Manager for callbacks
//      Location -> Where in key list was the key to be modified
//                  ( -1 if new key creation )
//
//  Builds Screen and sets up environment for callbacks.
//
public
FS_Service_Key_Constructor (Service_Key InKey,
                        File_System_Manager Parent,
                        int Location )
{
    // Create Window for display and set layout
    super ( "Modify/Create File System Key" ) ;
    setLayout( new BorderLayout () ) ;

    // Create a new top panel for the window and set layout
    Panel top = new Panel () ,
    top.setLayout ( new FlowLayout () ) ,

    // Set the call back parent and list selection
    amParent   Parent ,
    ListLocation   Location ;

    // Create Text fields and fill in values
    UserName = new TextField ( 30 ) ;
    UserHost   new TextField ( 30 ) ;
    AppService = new TextField ( 30 ) ,
    AppDisplay   new TextField ( 30 ) ;

    if ( InKey != null )
    {
        // Modifying key
        UserName.setText ( InKey.getUserName () ) ;
        UserHost.setText ( InKey.getUserHost () ) ;
        AppService.setText ( InKey.getServiceRequest () ) ,
        AppDisplay.setText ( InKey.getAdditionalInfo () ) ,
    }
    else
    {
        // Creating Key
        String UserInfo [] = amParent.getUserInfo () ,
        UserName.setText ( UserInfo[0] ) ,
```

```
        UserHost.setText ( UserInfo[1] ) ,
    }

    // Create list of cities to choose from
    Lister = new Choice ( ) ;

    Lister.addItem ( "New York" ) ,
    Lister.addItem ( "San Francisco" ) ,
    Lister.addItem ( "Hong Kong" ) ;
    Lister.addItem ( "London" ) ;

    Lister.select ( 0 ) ;

    // User name and host are read only
    UserName.setEditable( false ) ;
    UserHost.setEditable( false ) ;

    // Add text panes and list to top panel
    top.add ( "Center", new Label ("User Name:") ) ;
    top.add ( "Center", UserName ) ;
    top.add ( "Center", new Label ("User Host:") ) ,
    top.add ( "Center", UserHost ) ;
    top.add ( "Center", new Label ("File System Name:") ) ,
    top.add ( "Center", AppService ) ;
    top.add ( "Center", new Label ("File System Display Field:") ) ,
    top.add ( "Center", AppDisplay ) ,
    top.add ( "Center", new Label ("Location of File System") ) ;
    top.add ( "Center", Lister ) ;

    // Create a new bottom panel and add buttons to it
    Panel bottom = new Panel () ;
    OK_but   new Button("OK") ;
    Cancel_but   new Button("Cancel") ;
    bottom.add( OK_but ) ;
    bottom.add( Cancel_but ) ,

    // Add panles to window, set size and display window
    add("Center", top ) ;
    add("South", bottom ) ,
    resize( 300, 300 ) ,
    setResizable( false ) ;
    show () ,
}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean
handleEvent ( Event evt )
{
    // Determine if window event requires system action
    if ( evt.target.equals ( OK_but ) )
    {
        // Make a new key form input values and save with parent
        Service_Key SKey = new Service_Key () ;
        SKey.setUserName ( UserName.getText () ) ;
        SKey.setUserHost ( UserHost.getText () ) ;
        SKey.setServiceRequest ( AppService.getText () ) ,
        SKey.setAdditionalInfo ( AppDisplay.getText () ) ,
        byte Array []   SetBytefield( Lister.getSelectedIndex () ) ;
        SKey.setKey ( Array ) ;
        SKey.setHostPort ( UNS_API.Well_Known_Port ) ,
```

```
        amParent.Change_Callback ( SKey, ListLocation ) ,
        dispose () ;
        return true ;
    }
    else if ( evt.target.equals ( Cancel_but ))
    {
        // Cancel hit so discard information
        dispose () ;
        return true ,
    }
    else if ( evt.id == Event.WINDOW_DESTROY )
    {
        // Mian window X hit so discard info
        dispose () ,
        return true ;
    }
    // Or allow parent to handle event
    return super.handleEvent( evt ) ,
}

////////////////////////////////////////////
//  SetBytefield
//
//  Arguments
//      CityChoice -> fill in byte field with info
//              appropriate to a particulat city.
//              0   hpcims17
//              1 = hpcims18
//              2 = hpcims19
//              3   hpcims20
//
//  Returns
//      byte [] -> service key bytefiled with info
//              that indicates the appropriate host
//
private byte []
SetBytefield ( int CityChoice )
{
    // Strings that indicate Host and Basepath
    String Host ;
    String BasePath ,
    // Port on FSS server to connect to
    int    FSSPort   Desktop_Manager.FSSHostPort ,

    // Use city choice to set host and path info
    switch ( CityChoice )
    {
        case 0
            Host   new String ( "hpcims17.isc.rit.edu" ) ,
            BasePath   new String ( "/home/stu5/kam3011/home17" ) ,
            break ,
        case 1
            Host = new String ( "hpcims18.isc.rit.edu" ) ,
            BasePath   new String ( "/home/stu5/kam3011/home18" ) ,
            break ,
        case 2 .
            Host = new String ( "hpcims19.isc.rit.edu" ) ,
            BasePath   new String ( "/home/stu5/kam3011/home19" ) ,
            break ,
        default
            Host = new String ( "hpcims20.isc.rit.edu" ) ,
            BasePath   new String ( "/home/stu5/kam3011/home20" ) ,
            break ,
    }

    // Stream data into byte array
    ByteArrayOutputStream Byter   new ByteArrayOutputStream () ,
    DataOutputStream IODev   new DataOutputStream ( Byter ) ,
```

```
        try
        {
            IODev.writeUTF ( Host ) ;
            IODev.writeInt ( FSSPort ) ,
            IODev.writeUTF ( BasePath ) ,
        }
        catch ( IOException e )
        {
            return null ;
        }

        // return finilled in array
        return Byter.toByteArray() ;
    }
}
```

Login_Manager

```
/*****************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*****************************************************************************/

/*****************************************************************************
*   Class Login_Manager
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Manages the status of the system user (logged in or out) and maintains
*   Appropriate UNS API for the current user.
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Imports
*/
import java.io.* ;
import java.net.* ,
/**
*   Custom Imports
*/
import Keys.* ;
import UNSServer.* ;
import UNSUtils.* ,
import Desktop.* ,

/**
*   Class Definition
*/
public
class Login_Manager
{
    ////////////////////////////////////////////////////////
    //Private Member Variables
    //
    // API -> UNS_API used to communicate with the UNS to
    //          Log user in and out
    private
    UNS_API API   null ;
    // Parent -> Desktop Manager (Parent) for use in call backs
    //          after system events
    private
    Desktop_Manager theDesktop = null ,

    ////////////////////////////////////////////////
    //  Constructor
    //
```

```
//  Arguments:
//      inAPI -> the API that this object will use to
//               communicate with the UNS
//      intheDesktop -> Desktop Manager (Parent) for use in call backs
//               after system events
//
//  set up the callbacks and wait for commands
public
Login_Manager ( UNS_API inAPI, Desktop_Manager intheDesktop)
{
    // Store reference objects for future use.
    API = inAPI ,
    theDesktop   intheDesktop ,
}

////////////////////////////////////////////
//  Login_User
//
//  Arguments:
//      none
//
//  Returns:
//      Boolean   true
//
//  Creates a login info screen and returns
//
public boolean
Login_User ()
{
    new LoginWindow ( this ) ,
    return true ;
}

////////////////////////////////////////////
//  Logout_User
//
//  Arguments:
//      none
//
//  Returns:
//      Boolean   sucess of logout
//
//  Uses API to logout user and returns it to
//  the main desktop
//
public boolean
Logout_User ()
{
    boolean sucess   API.Logout_User ( ) ;
    theDesktop.Logout_Callback ( sucess ) ,
    return sucess ,
}

////////////////////////////////////////////
//  LoginCallBack
//
//  Arguments:
//      User_Name   Name user used to login
//      User_Pass = Password user entered
//
//  used by Login Window to initiate a login with
//  information entered by the user.
//
public void
LoginCallBack ( String User_Name, String User_Pass )
{
    // If either (or both) strings are empty,
    if (( User_Name == null ) || ( User_Pass == null ))
```

```
        {
            // Cancel or Kill
            // invalidate the login
            theDesktop.Login_Callback( false, null ) ;
        }
        else
        {
            // OK Button
            // Attempt to login
            boolean sucess   API.Login_User ( User_Name, User_Pass ) ,
            theDesktop.Login_Callback( sucess, User_Name ) ;
        }
    }
}
```

LoginWindow

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class LoginWindow
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows user to enter his/her user anme and password for use in
*   logging into the system
*
*       Extends Frame (for display attributes and functionality)
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Imports
*/
import java.awt.* ,

/**
*   Class Definition
*/
public
class LoginWindow extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> OK Button on Screen
    private
    Button OK_but   null ,
    // OK_but -> OK Button on Screen
    private
    Button Cancel_but   null ,
    // Manager -> Parent for use in callbacks
    private
    Login_Manager Manager   null ;
    // UserName -> User entry text area for user name
    private
    TextField UserName = null ;
    // UserPass -> User entry text area for user password
    private
    TextField UserPass   null ;
```

```
//////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      inManager -> Login Manager for callbacks
//
//  Builds Screen and sets up environment for callbacks.
//
public
LoginWindow ( Login_Manager inManager )
{
    // Call Parents constructor
    super ( "Login Window" ) ;
    setLayout( new BorderLayout () ) ,

    // Create new panels for window
    Panel top   new Panel () ;
    Panel bottom   new Panel () ,

    // Create text enty fields and add to top panel
    UserName   new TextField ( 40 ) ,
    UserPass   new TextField ( 40 ) ;
    UserPass.setEchoCharacter ( '*' ) ;
    UserName.setEditable(true) ,
    UserPass.setEditable(true) ;
    top.add ( new Label ("User Name:", Label.LEFT) ) ,
    top.add ( UserName ) ;
    top.add ( new Label ("User Password:", Label.LEFT) ) ;
    top.add ( UserPass ) ;

    // Create buttons and add to bottom panel
    OK_but   new Button("OK") ;
    Cancel_but = new Button("Cancel") ,
    bottom.add( OK_but ) ;
    bottom.add( Cancel_but ) ;

    // Store parent for callbacks
    Manager  inManager ;

    // Add panles to window, set size, attributes and show
    add("Center", top ) ;
    add("South", bottom ) ;
    resize(370, 170) ,
    setResizable( false ) ,
    show () ;
}

//////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean
handleEvent ( Event evt )
{
    // Look for specific events
    if ( evt.target.equals ( OK_but ) )
    {
        // OK Button Clicked
        // Get user name and password eneterd and call parent
        String NameString   UserName.getText () ;
        String PassString   UserPass.getText () ;
        Manager.LoginCallBack ( NameString, PassString ) ,
```

```
            dispose () ;
            return true ;
        }
        else if ( evt.target.equals ( Cancel_but ))
        {
            // Cancel Button Clicked
            // Call parent with failed login
            Manager.LoginCallBack ( null, null ) ,
            dispose () ;
            return true ,
        }
        else if ( evt.id == Event.WINDOW_DESTROY )
        {
            // X Button Clicked
            // Call parent with failed login
            Manager.LoginCallBack ( null, null ) ;
            dispose () ,
            return true ;
        }
        // Or let parent handle event
        return super.handleEvent( evt ) ;
    }
}
```

MainDesktop

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class MainDesktop
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Main control window for desktop environment.  Spawns User_Actions
*   and System_Actions windows to obtain more information from the user
*   as to a particular request.  Passes all requests to the Desktop_Manager
*   for further processing.
*
*   Extends Frame (for display attributes and functionality)
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Imports
*/
import java.awt.* ;

/**
*   Class Definition
*/
public
class MainDesktop extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // TextAr -> System status and feedback window
    private
    TextArea TextAr   null ,
    // UButton -> User Action button
    private
    Button UButton = null ,
    // SButton -> System Action button
    private
    Button SButton = null ;
    // UScreen -> Display window of possible User Actions
    private
    User_Actions UScreen   null ;
    // SScreen -> Display window of possible System Actions
```

```
private
System_Actions SScreen = null ,
// dmParent -> Parent to send user actions back to
private
Desktop_Manager dmParent = null ,

//////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      Parent -> Desktop Manager for callbacks
//
//  Builds Screen and sets up environment for callbacks.
//
public
MainDesktop ( Desktop_Manager Parent )
{
    // Make window and set layout
    super ( "Location Independant Desktop" ) ,
    setLayout( new BorderLayout () ) ,

    // Create a new panel and set layout
    Panel top   new Panel () ;
    top.setLayout ( new BorderLayout () ) ,

    // Create a text area for feedback and set up the initial text
    // also add to top panel
    TextAr   new TextArea("System Activity\n", 15, 5) ;
    TextAr.setEditable ( false ) ;
    TextAr.appendText ( "------------------------\n" ) ,
    top.add ("Center", TextAr ) ,

    // Create a new panel and set layout
    Panel bottom   new Panel () ,
    bottom.setLayout ( new FlowLayout () ) ,

    // Create buttons and add them to the bottom panel
    UButton = new Button("User Actions") ;
    SButton = new Button("System Actions") ,
    bottom.add( UButton ) ,
    bottom.add( SButton ) ;


    // Add panels to window, set window attributes and display
    add("Center", top ) ;
    add("South", bottom ) ,
    resize(300, 200) ,
    setResizable( false ) ,
    show () ;

    // Call back parent for system actions
    dmParent   Parent ;
}

//////////////////////////////////////////////
//  SubScreenAction
//
//  Arguments:
//      Action -> The sub screen button that initiated the action
//      Value -> the specific request being made by the sub screen
//
//  Uses Action and Value to determine if and what requests are being
//  made by the user through the sub screen (User_Actions and System_Actions)
//
public void
SubScreenAction ( int Action, int Value )
{
    // Major Action
```

```
switch ( Action )
{
    // Ok on User_Actions subscreen
    case User_OK :
        UScreen   null ,
        // Value from action (User Actions)
        switch ( Value )
        {
            // Login User
            case User_Value_Login
                TextAr.appendText ( "Login...." ) ;
                dmParent.Action_Center ( Desktop_Manager.LOGIN ) ,
                break ;
            // Logout User
            case User_Value_Logout
                TextAr.appendText ( "Logout...." ) ,
                dmParent.Action_Center ( Desktop_Manager.LOGOUT ) ,
                break ,
            // Start a new App
            case User_Value_View_App
                TextAr.appendText ( "Start Application.\n" ) ;
                dmParent.Action_Center ( Desktop_Manager.LIST_APPS ) ,
                break ;
            // View File Systems
            case User_Value_View_Fil
                TextAr.appendText ( "List File Systems.\n" ) ;
                dmParent.Action_Center ( Desktop_Manager.LIST_FILES ) ,
                break ;
            // Change Configuration
            case User_Value_Set_Config
                TextAr.appendText ( "Set Configuration.\n" ) ;
                dmParent.Action_Center ( Desktop_Manager.SET_CONFIG ) ;
                break ,
            default :
                break ,
        }
        break ;
    // User hit Canel ( User_Actions )
    case User_Cancel :
        UScreen   null ;
        break ;
    // User hit OK ( System_Actions )
    case System_OK
        SScreen = null ;
        switch ( Value )
        {
            // Shut down the desktop
            case Sys_Value_ShutDown :
                TextAr.appendText ( "System Shutdown.\n" ) ;
                dmParent.Action_Center ( Desktop_Manager.SHUTDOWN ) ;
                break ,
            // Display System Status
            case Sys_Value_SystemStatus
                TextAr.appendText ( "System Status.\n" ) ;
                dmParent.Action_Center ( Desktop_Manager.SYSTEM_STAT ) ,
                break ;
            // Modify FSS Keys
            case Sys_Value_ModifyFile
                TextAr.appendText ( "Modify File Heirarchy.\n" ) ;
                dmParent.Action_Center ( Desktop_Manager.MODIFY_FILE ) ,
                break ;
            // Modify Application Keys
            case Sys_Value_ModifyApps :
                TextAr.appendText ( "Modify Application Heirarchy.\n" ) ;
                dmParent.Action_Center ( Desktop_Manager.MODIFY_APPS ) ,
                break ;
            default
                break ,
```

```
            }
            break ;
        // User hit Canel ( System_Actions )
        case System_Cancel :
            SScreen = null ;
            break ,
        default :
            break ,
    }

}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean
handleEvent ( Event evt )
{
    // Look for specific events
    if ( evt.target.equals ( UButton ) )
    {
        //User Action
        if ( UScreen == null )
        {
            UScreen   new User_Actions ( dmParent.IsLoggedIn (), this ) ,
        }
        else
        {
            UScreen.show () ;
        }
    }
    else if ( evt.target.equals ( SButton ) )
    {
        //System Action
        if ( SScreen == null )
        {
            SScreen   new System_Actions ( dmParent.IsLoggedIn (), this ) ;
        }
        else
        {
            SScreen.requestFocus () ;
        }
    }

    switch ( evt.id )
    {
        // X on Frame
        case Event.WINDOW_DESTROY:
        {
            TextAr.appendText ( "Cannot close main window.\n" ) ,
            return true ,
        }
    }
    return true ;
}

////////////////////////////////////////////
//  DisplayMessage
//
//  Arguments
//      Message -> String containing message to be displayed
//              on the main desktop window text box
```

```
    //
    //  Displays a message on the main window text box
    //      (Adds a carriage return at the end of the string)
    //
    public void
    DisplayMessage ( String Message )
    {
        TextAr.appendText ( Message + "\n" ) ,
    }

    ////////////////////////////////////////////
    //  UpdateColors
    //
    //  Arguments
    //      Foreground -> desired color of foreground
    //              of main window
    //      Background -> desired color of background
    //              of main window
    //
    //  Attempt to set the window colors
    //      note: doesn't work!!!! >B^|
    //
    public void
    UpdateColors ( Color Foreground, Color Background )
    {
        super.setForeground( Foreground ) ;
        super.setBackground( Background ) ;
        super.repaint () ,
    }

    ////////////////////////////////////////////////////
    //Public Member Variables
    //
    //
    //  Folowing used in SubScreenAction for requesting various
    //  actions:
    public static final
    int     User_OK = 1 ;
    public static final
    int     User_Cancel   2 ,
    public static final
    int     System_OK   3 ;
    public static final
    int     System_Cancel   4 ,
    public static final
    int     User_Value_Login   1 ,
    public static final
    int     User_Value_Logout   2 ;
    public static final
    int     User_Value_View_App   3 ,
    public static final
    int     User_Value_View_Fil   4 ;
    public static final
    int     User_Value_Set_Config   5 ,
    public static final
    int     Sys_Value_ShutDown   1 ,
    public static final
    int     Sys_Value_SystemStatus   2 ,
    public static final
    int     Sys_Value_ModifyFile   3 ;
    public static final
    int     Sys_Value_ModifyApps = 4 ,
}
```

Service_Key_Constructor

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class Service_Key_Constructor
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows the user to modify or create a UNS service key.
*
*   Derived from Frame class
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Import(s)
*/
import java.awt.* ,
import java.io.* ,
/**
*   Custom Imports
*/
import UNSUtils.* ;
import Desktop.* ;
import Keys.* ,

/**
*   Class Definition
*/
public
class Service_Key_Constructor extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> Modify Button on Screen
    private
    Button OK_but = null ,
    // Cancel_but -> Cancel button on screen
    private
    Button Cancel_but   null ,
    // amParent -> Object to whom all call backs will be made
    private
    Application_Manager amParent   null ;
```

```
// UserName -> Text field that shows the user name in the service key
private
TextField UserName = null ;
// UserHost -> Text field that shows the user's host name in the service key
private
TextField UserHost = null ;
// AppService -> Text field that shows Application key name
private
TextField AppService   null ;
// AppService -> Text field that shows Application key display information
private
TextField AppDisplay   null ,
// ListLocation -> Location of FSS Service key that the user has
//      chosen to modify ( -1 if creating new )
private
int ListLocation = 0 ,

//////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InKey -> Service key of FSS service key to be modified
//                  ( null if new key creation )
//      Parent -> Application Manager for callbacks
//      Location -> Where in key list was the key to be modified
//                  ( -1 if new key creation )
//
//  Builds Screen and sets up environment for callbacks.
//
public
Service_Key_Constructor (Service_Key InKey,
                         Application_Manager Parent,
                         int Location )
{
    // Create Window for display and set layout
    super ( "Modify/Create Application Key" ) ,
    setLayout( new BorderLayout () ) ,

    // Create a new top panel for the window and set layout
    Panel top = new Panel () ,
    top.setLayout ( new FlowLayout () ) ,

    // Set the call back parent and list selection
    amParent = Parent ;
    ListLocation   Location ,

    // Create Text fields and fill in values
    UserName   new TextField ( 30 ) ,
    UserHost   new TextField ( 30 ) ;
    AppService   new TextField ( 30 ) ;
    AppDisplay = new TextField ( 30 ) ,

    if ( InKey != null )
    {
        UserName.setText ( InKey.getUserName () ) ,
        UserHost.setText ( InKey.getUserHost () ) ,
        AppService.setText ( InKey.getServiceRequest () ) ,
        AppDisplay.setText ( InKey.getAdditionalInfo () ) ;
    }
    else
    {
        String UserInfo [] = amParent.getUserInfo () ;
        UserName.setText ( UserInfo[0] ) ;
        UserHost.setText ( UserInfo[1] ) ,
    }

    // User name and host are read only
    UserName.setEditable( false ) ,
```

```
    UserHost.setEditable( false ) ;

    // Add text panes and list to top panel
    top.add ( "Center", new Label ("User Name:") ) ,
    top.add ( "Center", UserName ) ;
    top.add ( "Center", new Label ("User Host:") ) ;
    top.add ( "Center", UserHost ) ;
    top.add ( "Center", new Label ("Application Name:") ) ;
    top.add ( "Center", AppService ) ;
    top.add ( "Center", new Label ("Application Display Field:") ) ,
    top.add ( "Center", AppDisplay ) ,

    // Create a new bottom panel and add buttons to it
    Panel bottom = new Panel () ,
    OK_but   new Button("OK") ;
    Cancel_but   new Button("Cancel") ,
    bottom.add( OK_but ) ;
    bottom.add( Cancel_but ) ,

    // Add panels to window, set size and display window
    add("Center", top ) ;
    add("South", bottom ) ;
    resize( 300, 270 ) ;
    setResizable( false ) ,
    show () ,
}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean handleEvent ( Event evt )
{
    // Determine if window event requires system action
    if ( evt.target.equals ( OK_but ) )
    {
        // Make a new key from input values and save with parent
        Service_Key SKey   new Service_Key () ,
        SKey.setUserName ( UserName.getText () ) ;
        SKey.setUserHost ( UserHost.getText () ) ;
        SKey.setServiceRequest ( AppService.getText () ) ;
        SKey.setAdditionalInfo ( AppDisplay.getText () ) ,
        byte Array []   { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } ,
        SKey.setKey ( Array ) ;
        SKey.setHostPort ( UNS_API.Well_Known_Port ) ;
        amParent.Change_Callback ( SKey, ListLocation ) ,
        dispose () ;
        return true ;
    }
    else if ( evt.target.equals ( Cancel_but ))
    {
        // Cancel hit so discard information
        dispose () ,
        return true ,
    }
    else if ( evt.id == Event.WINDOW_DESTROY )
    {
        // frame X hit so discard information
        dispose () ;
        return true ;
    }
    // Or allow parent to handle event
    return super.handleEvent( evt ) ,
```

```
    }
}
```

User_Actions

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class User_Actions
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows the user to select from a list of user actions.
*
*   Derived from Prame class
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Import(s)
*/
import java.awt.* ,

/**
*   Class Definition
*/
public
class User_Actions extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> OK Button on Screen
    private
    Button OK_but   null ;
    // Cancel_but -> Cancel_but Button on Screen
    private
    Button Cancel_but   null ;
    // Lister -> List of possible user actions
    private
    List Lister = null ,
    // mdParent -> Object to make callbacks to
    private
    MainDesktop mdParent = null ;

    ////////////////////////////////////////////
    //  Constructor
    //
```

```
//  Arguments:
//      LogedIn -> Is the user logged in
//      Parent -> Callback object to request actions
//
//  Sets up window and prepares for callback action
//
public
User_Actions ( boolean LogedIn, MainDesktop Parent )
{
    // create window and set attributes
    super ( "User Actions" ) ,
    setLayout( new BorderLayout () ) ;

    // Create panel and set attributes
    Panel top = new Panel () ,

    // Set acllback parent
    mdParent   Parent ,

    // if logged in allow all actions (except login)
    // else just allow to login
    if ( !LogedIn )
    {
        Lister = new List ( ) ,
        Lister.addItem ( "Login New User" ) ,
        Lister.select ( 0 ) ;
    }
    else
    {
        Lister = new List (  ) ;
        Lister.addItem ( "Logout User" ) ,
        Lister.addItem ( "Start Application" ) ,
        Lister.addItem ( "View File Systems" ) ;
        Lister.addItem ( "Set Configuration" ) ,
        Lister.select ( 1 ) ,
    }

    // Add List to window and set list attributes
    top.add ( new Label ("Please Select:", Label.LEFT) ) ,
    top.add ( Lister ) ;

    // Create a new window and add buttons to it
    Panel bottom   new Panel () ,
    OK_but = new Button("OK") ;
    Cancel_but   new Button("Cancel") ,
    bottom.add( OK_but ) ;
    bottom.add( Cancel_but ) ;

    // Add panels to window, set window attributes and display
    add("Center", top ) ;
    add("South", bottom ) ;
    resize(200, 150) ;
    setResizable( false ) ,
    show () ;
}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean handleEvent ( Event evt )
{
    if ( evt.target.equals ( OK_but ) )
```

```
        {
            // OK Hit
            int ListChoice   Lister.getSelectedIndex () + 1 ;

            if ( Lister.countItems() != 1 )
            {
                ListChoice += 1 ,
            }
            mdParent.SubScreenAction(MainDesktop.User_OK,
                            ListChoice ) ,
            dispose () ,
        }
        else if ( evt.target.equals ( Cancel_but ))
        {
            // Cancel Hit
            mdParent.SubScreenAction(MainDesktop.User_Cancel, 0 ) ,
            dispose () ,
        }
        else if ( evt.id == Event.WINDOW_DESTROY )
        {
            // window frame X hit
            mdParent.SubScreenAction(MainDesktop.System_Cancel, 0 ) ,
            dispose () ,
        }
        return true ,
    }
}
```

System_Actions

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class System_Actions
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Allows the user to select from a list of system actions.
*
*   Derived from Frame class
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ,
/**
*   System Import(s)
*/
import java.awt.* ,

/**
*   Class Definition
*/
public
class System_Actions extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // OK_but -> OK Button on Screen
    private
    Button OK_but   null ;
    // Cancel_but -> Cancel_but Button on Screen
    private
    Button Cancel_but = null ;
    // Lister -> List of possible user actions
    private
    List Lister   null ,
    // mdParent -> Object to make callbacks to
    private
    MainDesktop mdParent = null ,

    ////////////////////////////////////////////
    //  Constructor
    //
```

```
//  Arguments:
//      LogedIn -> Is the user logged in
//      Parent -> Callback object to request actions
//
//  Sets up window and prepares for callback action
//
public
System_Actions ( boolean LogedIn, MainDesktop Parent )
{
    // create window and set attributes
    super ( "System Actions" ) ;
    setLayout( new BorderLayout () ) ,

    // Create panel and set attributes
    Panel top   new Panel () ;

    // Set acllback parent
    mdParent = Parent ;

    // if logged in allow all actions
    // else just allow to shutdown and see
    //  status.
    if ( LogedIn )
    {
        Lister   new List ( ) ;
        Lister.addItem ( "ShutDown" ) ;
        Lister.addItem ( "System Status" ) ,
        Lister.addItem ( "Modify File Keys" ) ,
        Lister.addItem ( "Modify Applications" ) ;
    }
    else
    {
        Lister   new List (  ) ;
        Lister.addItem ( "ShutDown" ) ,
        Lister.addItem ( "System Status" ) ;
    }

    // Add List to window and set list attributes
    Lister.select ( 1 ) ;
    top.add ( new Label ("Please Select:              ", Label.LEFT) ) ;
    top.add ( Lister ) ,

    // Create a new window and add buttons to it
    Panel bottom   new Panel () ,
    OK_but   new Button("OK") ;
    Cancel_but   new Button("Cancel") ,
    bottom.add( OK_but ) ;
    bottom.add( Cancel_but ) ,

    // Add panels to window, set window attributes and display
    add("Center", top ) ;
    add("South", bottom ) ,
    resize(200, 150) ;
    setResizable( false ) ,
    show () ,
}

////////////////////////////////////////////
//  handleEvent
//
//  Arguments
//      evt -> System generated Event
//
//  Parent designated handler of all system events
//  pertaining to this window.
//
public boolean handleEvent ( Event evt )
{
```

```
        if ( evt.target.equals ( Cancel_but ))
        {
            // Cancel Hit
            mdParent.SubScreenAction(MainDesktop.System_Cancel, 0 ) ,
            dispose () ,
        }
        if ( evt.target.equals ( OK_but ) )
        {
            // OK Hit
            int ListChoice   Lister.getSelectedIndex () + 1 ;
            mdParent.SubScreenAction(MainDesktop.System_OK,
                            ListChoice ) ,
            dispose () ;
        }

        if ( evt.id == Event.WINDOW_DESTROY )
        {
            // window frame X hit
            mdParent.SubScreenAction(MainDesktop.System_Cancel, 0 ) ,
            dispose () ;
        }
        return true ,
    }
}
```

User_Config_Manager

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class User_Config_Manager
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Manages the configuration of the system user and maintains
*   Appropriate UNS API calls for the current user config.
*/

/**
*   Package: Desktop
*       Package intended to be repository of all LID Desktop object
*/
package Desktop ;
/**
*   System Imports
*/
import java.io.* ,
import java.awt.* ,
import java.net.* ;

import Keys.* ,
/**
*   Custom Imports
*/
import UNSServer.* ,
import UNSUtils.* ,
import Desktop.* ;

/**
*   Class Definition
*/
public
class User_Config_Manager
{
    ////////////////////////////////////////////////////////
    //Private Member Variables
    //
    // API -> UNS_API used to communicate with the UNS to
    //          Log user in and out
    private
    UNS_API API = null ;
    // dmParent -> Desktop Manager (Parent) for use in call backs
    //          after system events
    private
```

```
Desktop_Manager dmParent   null ;
//  userConfig -> Manager of information for the user
private
Config_Manager userConfig = null ,

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      inAPI -> the API that this object will use to
//               communicate with the UNS
//      Parent -> Desktop Manager (Parent) for use in call backs
//               after system events
//
//  set up the callbacks and wait for commands
public
User_Config_Manager ( UNS_API inAPI, Desktop_Manager Parent )
{
    API = inAPI ,
    dmParent = Parent ,
    userConfig   API.Get_User_Configuration ( ) ;
}

////////////////////////////////////////////
//  Update_Config
//
//  Arguments:
//      none
//
//  Creates an config modify window and
//  displays it with the current config info
//
public void
Update_Config ()
{
    new Config_Window ( this,
        userConfig.GetForegroundColor(),
        userConfig.GetBackgroundColor() ) ,
}

////////////////////////////////////////////
//  UpdateColorsCallback
//
//  Arguments:
//      ColorArray -> Array of six ints that
//          contain the Red, Green, & Blue
//          (ordered) values for the fore/
//          background colors.
//
//  used by the config window as a callback
//  for updating user colors
//
public void
UpdateColorsCallback ( int ColorArray[] )
{
    Color Front   new Color ( ColorArray [0],
                              ColorArray [1],
                              ColorArray [2] ) ,
    Color Back    new Color ( ColorArray [3],
                              ColorArray [4],
                              ColorArray [5] ) ;

    userConfig.SetForegroundColor ( Front ) ,
    userConfig.SetBackgroundColor ( Back ) ;

    API.Set_User_Configuration ( userConfig ) ,
    dmParent.Update_Colors ( Front, Back ) ,
}
```

```
    ////////////////////////////////////////////
    //  UpdateColorsCallback
    //
    //  Arguments:
    //      none
    //
    //  returns the current user's foreground color
    //
    public Color
    GetForeground ()
    {
        Color Result   userConfig.GetForegroundColor ( ) ,
        return Result ;
    }

    ////////////////////////////////////////////
    //  UpdateColorsCallback
    //
    //  Arguments:
    //      none
    //
    //  returns the current user's background color
    //
    public Color
    GetBackground ()
    {
        Color Result = userConfig.GetBackgroundColor ( ) ;
        return Result ;
    }
}
```

APIs

File List:
EXE_API.java
AS_API.java
UNS_API.java

EXE_API

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*   Class EXE_API
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Provices an interface for executables to request information from
*   the main desktop and to provice a conduit for System out/err messages
*   to be displayed to the desktop's text ports
*/

/**
*   Package: DesktopUtils
*       Package intended to be repository of all LID interface objects
*/
package DesktopUtils ,
/**
*   System Imports
*/
import java.io.* ;
import java.util.* ,
/**
*   Custom Imports
*/
import Desktop.* ,
import UNSUtils.* ;

/**
*   Class Definition
*/
public
class EXE_API
{
    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
    //      none
    //
    //  No class variables needed yet
    //
    public
    EXE_API ()
    {
    }
```

```
//////////////////////////////////////////
// Get Heirarchy
//
//  Contact the desktop (execution manager) and obtain the
//  File System Service Keys (in a Heirarchy Manager)
//
public Vector
GetHeirarchy ( )
{
    try
    {
        // create a stream for output data
        DataOutputStream DataOut   new DataOutputStream ( System.out ) ,

        // Inform the desktop what request is being made
        DataOut.writeInt ( Execution_Manager.GETHEIRARCHY ) ;

        // Make sure that the desktop receives the request
        DataOut.flush () ;

        // Create a stream for input data
        DataInputStream DataIn = new DataInputStream ( System.in ) ;

        // Construct a new Heirarchy Manager from the input stream
        Heirarchy_Manager HMan   new Heirarchy_Manager ( DataIn ) ;

        Vector response   new Vector ( HMan.GetTotalIndexs () ) ;

        for ( int i   0 ; i < HMan.GetTotalIndexs () , i++ )
        {
            response.addElement( HMan.GetKeyAtIndex(i) ) ;
        }

        return response ,
    }
    catch ( IOException e )
    {
        // Ignore IO Errors
        return null ,
    }
}

//////////////////////////////////////////
// GetUserFSSHost
//
//  Contact the desktop (execution manager) and request
//  that it supply the (local) File System Server Host
//
public String
GetUserFSSHost ( )
{
    try
    {
        // create a stream for output data
        DataOutputStream DataOut = new DataOutputStream ( System.out ) ,

        // Inform the desktop what request is being made
        DataOut.writeInt ( Execution_Manager.GETFSSHost ) ;

        // Make sure that the desktop receives the request
        DataOut.flush () ;

        // create a stream for input data
        DataInputStream DataIn = new DataInputStream ( System.in ) ,

        String Host = new String ( DataIn.readUTF () ) ,
```

```
            return Host ,
        }
        catch ( IOException e )
        {
            // Ignore IO Errors
            return null ;
        }
    }

    //-----------------------------------------------
    // GetUserFSSPort
    //
    //  Contact the desktop (execution manager) and request
    //  that it supply the (local) File System Server Host
    //
    public int
    GetUserFSSPort ( )
    {
        try
        {
            // create a stream for output data
            DataOutputStream DataOut   new DataOutputStream ( System.out ) ;

            // Inform the desktop what request is being made
            DataOut.writeInt ( Execution_Manager.GETFSSPort ) ;

            // Make sure that the desktop receives the request
            DataOut.flush () ;

            // create a stream for input data
            DataInputStream DataIn   new DataInputStream ( System.in ) ;

            return DataIn.readInt () ,
        }
        catch ( IOException e )
        {
            // Ignore IO Errors
            return -1 ,
        }
    }

    //-----------------------------------------------
    // Send Message
    //
    //  Contact the desktop (execution manager) and request
    //  that it display a message to standard out
    //
    public void
    SendMessage ( String MessageOut )
    {
        try
        {
            // create a stream for output data
            DataOutputStream DataOut   new DataOutputStream ( System.out ) ;

            // Inform the desktop what request is being made
            DataOut.writeInt ( Execution_Manager.TEXTOUTPUT ) ,

            // Make sure that the desktop receives the request
            DataOut.flush () ;

            // Send the message to be displayed
            System.out.println ( MessageOut ) ,

            // Make sure that the desktop receives the message
            System.out.flush () ,
        }
        catch ( IOException e )
```

```
        {
            // Ignore IO Errors
        }
    }
}
```

AS_API

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*   Class AS_API
*
*   Jeff Harman
*
*   Date        Revision    Comment
*   4/4/97      1.0         Creation
*
*   Provices an interface for executables to request applications from
*   the local AS Server and to provice a conduit verification of user
*   requests to the AS Server
*/

/**
*   Package: ASUtils
*       Package intended to be repository of all AS Server interface objects
*/
package ASUtils ,

/**
*   System Imports
*/
import java.net.* ;
import java.io.* ,
/**
*   Custom Imports
*/
import Keys.* ,
import ASServer.* ,

/**
*   Class Definition
*/
public
class AS_API
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  Host -> DNS Address of the local AS Server
    private
    String Host = null ;
    //  Host_Conn -> The socket that connects to the local AS Server
    private
    Socket Host_Conn = null ;
    //  rawDataIn -> unformatted port reading data from source
    private
    InputStream rawDataIn   null ;
```

```
//  rawDataOut -> unformatted port writing data to source
private
OutputStream rawDataOut = null ,
//  DataIn -> port reading data from source
private
DataInputStream DataIn   null ,
//  DataOut -> port writing data to source
private
DataOutputStream DataOut   null ;
//  CKey -> Channel key used to verify connection
private
Channel_Key CKey   null ;

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      Local_Host -> local AS Server DNS Address
//
//  serts the local AS server DNS Address
//
public
AS_API ( String Local_Host )
{
    Host   new String ( Local_Host ) ,
}

////////////////////////////////////////////
//  Start_New_Service
//
//  Arguments:
//      SKey -> Service key that indicates the
//          application being requested
//
//  initiates a connection with the AS Server and establishes
//  the request being made (through the service key)
//
public boolean
Start_New_Service ( Service_Key SKey )
{
    try
    {
        // Connect to the local AS Server
        Host_Conn   new Socket ( Host, Well_Known_Port ) ,

        // Set up the data handling ports
        rawDataIn   Host_Conn.getInputStream () ;
        rawDataOut = Host_Conn.getOutputStream () ;
        DataIn   new DataInputStream(rawDataIn) ;
        DataOut   new DataOutputStream(rawDataOut) ,

        // Let the ASServer that a new service is being requested
        DataOut.writeInt ( ASClient.NEWSERVICE ) ;

        // Pack up the service key and send it out
        byte SKeyBuffer []   SKey.getKeyStream () ,
        DataOut.writeInt ( SKeyBuffer.length ) ;
        DataOut.write ( SKeyBuffer, 0 , SKeyBuffer.length ) ;

        // Get the response from the server
        boolean ReplyValue = DataIn.readBoolean () ;

        if ( ReplyValue )
        {
            // Passed startup

            // Get the channel key form the server
            int Length   DataIn.readInt () ,
```

```
            byte CKeyBuf []   new byte [ Length ] ;
            DataIn.read ( CKeyBuf, 0, Length ) ;
            CKey = new Channel_Key ( CKeyBuf ) ,

            // Let the client know that it was successful
            return true ;
        }
        else
        {
            // Failed startup
            Host_Conn   null ,
            rawDataIn   null ;
            rawDataOut   null ;
            DataIn   null ,
            DataOut   null ,
            CKey   null ;

            // Let the client know that it failed
            return false ;
        }
    }
    catch (IOException e)
    {
        Host_Conn   null ,
        rawDataIn   null ;
        rawDataOut   null ,
        DataIn   null ,
        DataOut   null ;
        CKey   null ,

        return false ;
    }
}

////////////////////////////////////////////
//  Get_Application
//
//  Arguments:
//      none
//
//  requests that the application be sent to the
//  client.  Returns the buffer that contains the
//  application.  Returns null if request failed.
//
public byte []
Get_Application ()
{
    try
    {
        // Request the Application
        DataOut.writeInt ( ASClient.GETAPPLICATION ) ,

        // Pack up the channel key and send it out
        // to verify the request
        byte CKeyBuf [] = CKey.getKeyStream () ,
        DataOut.writeInt ( CKeyBuf.length ) ;
        DataOut.write ( CKeyBuf, 0, CKeyBuf.length ) ;

        // Get server's response
        boolean Response   DataIn.readBoolean () ;

        if ( Response )
        {
            // Server granting request
            // So get the buffer and return it
            int Length   DataIn.readInt () ;

            byte ApBuffer []   new byte [Length] ,
```

```
            DataIn.readFully ( ApBuffer ) ,

            return ApBuffer ,
        }
        else
        {
            // Server not granting request
            // So close connection and return null
            Host_Conn   null ;
            rawDataIn   null ,
            rawDataOut   null ;
            DataIn   null ;
            DataOut   null ,
            CKey   null ,

            return null ,
        }
    }
    catch ( IOException e )
    {
        Host_Conn   null ;
        rawDataIn   null ;
        rawDataOut   null ,
        DataIn   null ;
        DataOut   null ,
        CKey   null ,

        return null ,
    }
}

////////////////////////////////////////////
//  Close_Service
//
//  Arguments:
//      none
//
//  requests that the server close the connection
//
//  returns true if sucessful or false if not.
//
public boolean
Close_Service ()
{
    try
    {
        // Tell server to close connection
        DataOut.writeInt ( ASClient.GETAPPLICATION ) ,

        // Write channel key to verify request
        byte CKeyBuf []   CKey.getKeyStream () ,
        DataOut.writeInt ( CKeyBuf.length ) ;
        DataOut.write ( CKeyBuf, 0, CKeyBuf.length ) ;

        // Don't care about server response
        //boolean Response   DataIn.readBoolean () ,

        // close connection
        Host_Conn = null ,
        rawDataIn = null ,
        rawDataOut = null ;
        DataIn   null ;
        DataOut   null ,
        CKey   null ,

        // Reply to user
        return true ;
```

```
        }
        catch ( IOException e )
        {
            Host_Conn   null ,
            rawDataIn   null ;
            rawDataOut   null ,
            DataIn   null ,
            DataOut   null ;
            CKey   null ;

            return true ,
        }
    }

    ////////////////////////////////////////////////////
    //Public Member Variables
    //
    // Well_Known_Port -> Connection port for the AS
    public final static
    int Well_Known_Port   1997 ,
}
```

UNS_API

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    UNS API Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    3/26/97     1.0         Creation
*
*   The UNS API is implemented as an object with a set of
*   public member functions that carry out specific actions
*   for the client.  These actions are dependent on the state
*   of the API object (in some cases) or may be used independent
*   of the object status.  The UNS API is intended to automate
*   and simplify communication to the UNS Servers by providing
*   an interface to the server and thus insulating the user of
*   the API from interaction specifics.
*/

/**
*   Package: UNSUtils
*       Package intended to be repository of all UNS Server interface
*       and utility objects
*/
package UNSUtils ,

/**
*   System Imports
*/
import java.io.* ;
import java.net.* ,
/**
*   Custom Imports
*/
import Keys.* ,

/**
*   Class Definition
*/
public
class UNS_API
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  Primary_Host -> DNS Address of the local UNS Server
    private String
    Primary_Host   null ;
    //  Connection_Port -> The socket that connects to the local UNS Server
    private
```

```
Socket Connection_Port = null ,
//  rawDataIn -> unformatted port reading data from source
private
InputStream rawDataIn = null ;
//  rawDataOut -> unformatted port writing data to source
private
OutputStream rawDataOut   null ;
//  DataIn -> port reading data from source
private
DataInputStream DataIn   null ;
//  DataOut -> port writing data to source
private
DataOutputStream DataOut   null ;
// User_Logged_in -> Indicates if the API has a currently
//      logged in user connected to a UNS Server
private boolean
User_Logged_In   false ;

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      Host -> local UNS Server DNS Address
//
//  sets the local UNS server DNS Address and
//  object state
//
public
UNS_API (String Host)
{
        User_Logged_In = false ,
        Connection_Port   null ,
        rawDataIn = null ;
        rawDataOut   null ,

        Primary_Host   new String ( Host ) ,
}

////////////////////////////////////////////
//  Login_User
//
//  Arguments:
//      User_Name -> Name of the user attempting to log in
//          must contain and '@' symbol that separates the
//          name into two parts, right -> User primary UNS Host
//          left -> User Name
//      User_Password -> Password for the user attempting to
//          log into the uns server
//
//  returns true if UNS Server accepts request
//      (communication channel is established)
//  returns false if server rejects request
//      (no channel is established)
public boolean
Login_User ( String User_Name, String User_Password )
{
    try
    {
        // Connect to host
        Connection_Port   new Socket ( Primary_Host, Well_Known_Port ) ;

        // Build data ports from connection
        rawDataIn   Connection_Port.getInputStream () ;
        rawDataOut = Connection_Port.getOutputStream () ;
        DataIn = new DataInputStream(rawDataIn) ;
        DataOut = new DataOutputStream(rawDataOut) ,

        // Give server login information
```

```
        DataOut.writeInt ( Login ) ,
        DataOut.writeUTF ( User_Name ) ;
        DataOut.writeUTF ( User_Password ) ,

        // Get server response
        User_Logged_In   DataIn.readBoolean () ,
        return User_Logged_In ;
    }
    catch ( UnknownHostException e )
    {
        User_Logged_In   false ,
        Connection_Port   null ,
        rawDataIn = null ;
        rawDataOut   null ;
        DataIn   null ,
        DataOut = null ,

        return User_Logged_In ,
    }
    catch ( IOException e )
    {
        User_Logged_In   false ,
        Connection_Port   null ;
        rawDataIn = null ,
        rawDataOut   null ,
        DataIn   null ;
        DataOut   null ,

        return User_Logged_In ;
    }
}

////////////////////////////////////////////
//  Logout_User
//
//  Arguments:
//      none
//
//  returns true if UNS Server accepts request to logout
//  returns false if server rejects request
//
//  connection is broken in either case
//
//  Requires that the a user has logged in and not logged out
//
public boolean
Logout_User ( )
{
    try
    {
        if ( User_Logged_In )
        {
            // Write logout to host
            DataOut.writeInt ( Logout ) ;

            // Get result from server
            boolean Result    DataIn.readBoolean () ,

            // Kill Connection
            Connection_Port   null ,
            rawDataIn   null ,
            rawDataOut   null ;
            DataIn = null ;
            DataOut = null ;

            User_Logged_In   false ,

            return Result ;
```

```
        }
        else
        {
            Connection_Port   null ;
            rawDataIn = null ,
            rawDataOut   null ;
            DataIn   null ;
            DataOut   null ,

            return true ;
        }
    }
    catch ( IOException e )
    {
        User_Logged_In   false ,
        Connection_Port = null ,
        rawDataIn = null ;
        rawDataOut   null ,
        DataIn = null ;
        DataOut   null ;
        return false ,
    }
}

////////////////////////////////////////////
//  Get_User_Heirarchy_Apps
//
//  Arguments:
//      none
//
//  request the server send the list of application
//  service keys
//
//  return valid heirarchy_manager if sucessful or
//  null if not.
//
//  if failure then connection to host is probably
//  lost, login will be necessaary.
//
//  Requires that the a user has logged in and not logged out
//
public Heirarchy_Manager
Get_User_Heirarchy_Apps ()
{
    Heirarchy_Manager Result   null ,

    try
    {
        if ( User_Logged_In )
        {
            DataOut.writeInt ( Get_Heirarchy_Apps ) ,

            Result   new Heirarchy_Manager ( DataIn ) ;
        }
    }
    catch ( IOException e )
    {
        User_Logged_In = false ,
        Connection_Port   null ,
        rawDataIn = null ;
        rawDataOut   null ,
        DataIn   null ,
        DataOut = null ,

        Result   null ,
    }

    return Result ;
```

```
}

////////////////////////////////////////////
//  Get_User_Heirarchy_File
//
//  Arguments:
//      none
//
//  request the server send the list of file system
//  service keys
//
//  return valid heirarchy_manager if sucessful or
//  null if not.
//
//  if failure then connection to host is probably
//  lost, login will be necessaary.
//
//  Requires that the a user has logged in and not logged out
//
public Heirarchy_Manager
Get_User_Heirarchy_File ()
{
    Heirarchy_Manager Result   null ;

    try
    {
        if ( User_Logged_In )
        {
            DataOut.writeInt ( Get_Heirarchy_File ) ,

            Result - new Heirarchy_Manager ( DataIn ) ;
        }
    }
    catch ( IOException e )
    {
        User_Logged_In   false ,
        Connection_Port = null ,
        rawDataIn   null ;
        rawDataOut   null ;
        DataIn   null ,
        DataOut   null ,

        Result   null ,
    }

    return Result ,
}

////////////////////////////////////////////
//  Set_User_Heirarchy_Apps
//
//  Arguments:
//      SaveManager -> Manager that contains the list
//          of service keys to be stored on the UNS Server
//
//  request the server receive and store a list of application
//  service keys
//
//  return true if sucessful or
//  false if not.
//
//  if failure then connection to host is probably
//  lost, login will be necessaary.
//
//  Requires that the a user has logged in and not logged out
//
public boolean
Set_User_Heirarchy_Apps ( Heirarchy_Manager SaveManager )
```

```
{
    boolean Result = false ;

    try
    {
        if ( User_Logged_In )
        {
            DataOut.writeInt ( Set_Heirarchy_Apps ) ;

            SaveManager.WriteHeirarchy ( DataOut ) ;

            Result   true ,
        }
    }
    catch ( IOException e )
    {
        User_Logged_In   false ;
        Connection_Port   null ,
        rawDataIn = null ,
        rawDataOut   null ;
        DataIn   null ,
        DataOut   null ;

        Result   false ,
    }

    return Result ,
}

////////////////////////////////////////////
//  Set_User_Heirarchy_File
//
//  Arguments:
//      SaveManager -> Manager that contains the list
//          of service keys to be stored on the UNS Server
//
//  request the server receive and store a list of File System
//  service keys
//
//  return true if sucessful or
//  false if not.
//
//  if failure then connection to host is probably
//  lost, login will be necessaary.
//
//  Requires that the a user has logged in and not logged out
//
public boolean
Set_User_Heirarchy_File ( Heirarchy_Manager SaveManager )
{
    boolean Result   false ;

    try
    {
        if ( User_Logged_In )
        {
            DataOut.writeInt ( Set_Heirarchy_File ) ;

            SaveManager.WriteHeirarchy ( DataOut ) ,

            Result   true ,
        }
    }
    catch ( IOException e )
    {
        User_Logged_In   false ;
        Connection_Port   null ;
        rawDataIn = null ;
```

```
        rawDataOut = null ;
        DataIn = null ,
        DataOut = null ;

        Result   false ,
    }

    return Result ,
}

////////////////////////////////////////////
//  Get_User_Heirarchy_File
//
//  Arguments:
//      none
//
//  request the server send the user configuration
//
//  return valid Config_Manager if sucessful or
//  null if not.
//
//  if failure then connection to host is probably
//  lost, login will be necessaary.
//
//  Requires that the a user has logged in and not logged out
//
public Config_Manager
Get_User_Configuration ()
{
    Config_Manager Result   null ,

    try
    {
        if ( User_Logged_In )
        {
            DataOut.writeInt ( Get_Config ) ,

            Result   new Config_Manager ( DataIn ) ;
        }
    }
    catch ( IOException e )
    {
        User_Logged_In   false ,
        Connection_Port = null ;
        rawDataIn = null ;
        rawDataOut = null ,
        DataIn   null ;
        DataOut   null ,

        Result   null ,
    }

    return Result ,
}

////////////////////////////////////////////
//  Set_User_Configuration
//
//  Arguments:
//      SaveConfig -> Manager that contains the configuration
//          to be stored on the UNS Server
//
//  request the server receive and store the user configuration
//
//  return true if sucessful or
//  false if not.
//
//  if failure then connection to host is probably
```

```
//  lost, login will be necessaary.
//
//  Requires that the a user has logged in and not logged out
//
public boolean
Set_User_Configuration (Config_Manager SaveConfig)
{
    boolean Result   false ,

    try
    {
        if ( User_Logged_In )
        {
            DataOut.writeInt ( Set_Config ) ,

            SaveConfig.WriteStream ( DataOut ) ;

            Result   true ,
        }
    }
    catch ( IOException e )
    {
        User_Logged_In   false ,
        Connection_Port = null ,
        rawDataIn = null ;
        rawDataOut   null ,
        DataIn   null ,
        DataOut = null ;

        Result   false ,
    }

    return Result ;
}

////////////////////////////////////////////
//  Verify_Service_Key
//
//  Arguments:
//      Test_Key -> Key to verified as valid
//
//  If the user is logged in:
//      contacts the connected UNS Server and requests
//      the key be verified
//  If the user is not logged in:
//      conatcts the UNS server specified in the service
//      key to verify the key
//
public boolean
Verify_Service_Key ( Service_Key Test_Key )
{
    boolean Result   false ;

    try
    {
        if ( User_Logged_In )
        {
            // User logged in so use connection
            // and request verification of key
            DataOut.writeInt ( Verify_Key ) ;

            byte Stream [] = Test_Key.getKeyStream () ;

            DataOut.writeInt ( Stream.length ) ,

            DataOut.write ( Stream, 0, Stream.length ) ;

            Result = DataIn.readBoolean () ,
```

```
            return Result ;
        }
        else
        {
            // User not logged in so connect to server
            // and request verification of key

            // Connect to server
            Connection_Port
                new Socket ( Test_Key.getUserHost(), Test_Key.getHostPort() ) ,

            // Set data ports
            rawDataIn = Connection_Port.getInputStream () ;
            rawDataOut   Connection_Port.getOutputStream () ,
            DataIn   new DataInputStream(rawDataIn) ;
            DataOut = new DataOutputStream(rawDataOut) ,

            // Request verification
            DataOut.writeInt ( Verify_Key ) ,

            // Pack up key and send
            byte Stream []   Test_Key.getKeyStream () ,
            DataOut.writeInt ( Stream.length ) ,
            DataOut.write ( Stream, 0, Stream.length ) ;

            // Get result from server
            Result   DataIn.readBoolean () ;

            // Close connection
            DataIn   null ;
            DataOut   null ;
            rawDataIn   null ,
            rawDataOut   null ;
            Connection_Port   null ,

            return Result ;
        }
    }
    catch ( UnknownHostException e )
    {
        User_Logged_In   false ,
        Connection_Port   null ,
        rawDataIn = null ;
        rawDataOut   null ,
        DataIn   null ,
        DataOut   null ;

        Result   false ,
    }
    catch ( IOException e )
    {
        User_Logged_In   false ,
        Connection_Port   null ,
        rawDataIn = null ;
        rawDataOut   null ;
        DataIn = null ,
        DataOut   null ,

        Result   false ;
    }

    return Result ,
}

//////////////////////////////////////////////////////
//Public Member Variables
//
```

```
    //  Requests to be made to server
    static final public
    int Login   0 ;
    static final public
    int Logout   1 ;
    static final public
    int Get_Heirarchy_Apps   3 ;
    static final public
    int Set_Heirarchy_Apps = 4 ,
    static final public
    int Get_Heirarchy_File   5 ;
    static final public
    int Set_Heirarchy_File   6 ,
    static final public
    int Get_Config   7 ;
    static final public
    int Set_Config   8 ,
    static final public
    int Verify_Key   9 ,
    // Server status of request values
    public static final
    int Local_User_Pass= 1 ,
    public static final
    int Local_User_Fail= 2;
    public static final
    int Non_Local_User   3 ;
    // Server request port
    public static final
    int Well_Known_Port   1968 ,
}
```

UNS Server

File List:
Mediator.java
User_Record.java
UNSClient.java
UNSCommManager.java
User_Database.java
BootStrap.java
UNSServer.java
Relay_Thread.java

Mediator

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    Mediator Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    3/26/97     1.0         Creation
*
*   Object that relays commands form individual client
*   connection threads to the user database.
*
*   Main access routine is synchronized to only allow 1 client in
*   at a time to eliminate confilicts
*/

/**
*   Package: UNSServer
*       Package intended to be repository of all UNSServer objects
*/
package UNSServer ;

/**
*   System Imports
*/
import java.lang.* ,
import java.net.* ;
import java.io.* ;
/**
*   Custom Imports
*/
import Keys.* ,
import UNSUtils.* ,

/**
*   Class Definition
*/
public
class Mediator
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // rawDataIn -> socet data input port
    private
    InputStream rawDataIn   null ,
    // rawDataOut -> socet data output port
    private
    OutputStream rawDataOut   null ,
```

```
// DataIn -> formatted data input port
private
DataInputStream DataIn = null ,
// DataOut -> formatted data output port
private
DataOutputStream DataOut   null ,
// Application -> Selector of type of Heirarchy request
//          that is being made
private static
int Application   1 ;
// File -> Selector of type of Heirarchy request
//          that is being made
private static
int File = 2 ;
// DataBase -> Respository of all information about users
private User_Database
DataBase = null ;

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      inDataBase -> the database that all user requests
//                        are satisfied from
//
//  Sets the dets the database name
public
Mediator (User_Database inDataBase)
{
    DataBase   inDataBase ,
}

////////////////////////////////////////////
//  New_Request
//
//  Arguments:
//      Request_Type -> the type of request being made
//      IO_Port -> The socket that converses with the client
//      User_Name -> Used for record selection and keying
//
//  Determine what request is being made then calls the
//  appropriate routine to handle it.
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
public synchronized
int New_Request (int Request_Type, Socket IO_Port, String User_Name ) throws IOException
{
    // Extract data ports from the raw socket
    rawDataIn   IO_Port.getInputStream () ;
    rawDataOut = IO_Port.getOutputStream () ,
    DataIn   new DataInputStream(rawDataIn) ;
    DataOut   new DataOutputStream(rawDataOut) ;

    // Determine what request has been made and handle it
    switch (Request_Type)
    {
        // Login a user
        case UNS_API.Login ·
                return Login_User ( User_Name ) ,
        // Logout a user
        case UNS_API.Logout
                return Logout_User ( User_Name )  ;
        // Get application service keys
        case UNS_API.Get_Heirarchy_Apps
                return Get_Heirarchy ( User_Name, Application ) ,
        // Set application service keys
        case UNS_API.Set_Heirarchy_Apps
```

```
                return Set_Heirarchy ( User_Name, Application ) ;
        // Get file system service keys
        case UNS_API.Get_Heirarchy_File :
                return Get_Heirarchy ( User_Name, File ) ,
        // Set file system service keys
        case UNS_API.Set_Heirarchy_File :
                return Set_Heirarchy ( User_Name, File ) ,
        // Get user configuration
        case UNS_API.Get_Config
                return Get_Config ( User_Name ) ,
        // Set user configuration
        case UNS_API.Set_Config
                return Set_Config ( User_Name ) ,
        // Verify a user service key
        case UNS_API.Verify_Key
                return Verify_Key ( ) ;
        default :
        // Invalid request
                return UNS_API.Local_User_Fail ,
    }
}

//////////////////////////////////////////////
//  Login_User
//
//  Arguments:
//      User_Name -> name of the user attempting to login
//
//  Attempts to login a user,
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
private int
Login_User ( String User_Name ) throws IOException
{
    // Is user name valid?
    int record = DataBase.GetUserIndex( User_Name ) ;
    if ( record == -1 )
    {
        // User Not Found
        DataOut.writeBoolean ( false ) ;
        return UNS_API.Local_User_Fail ,
    }
    else
    {
        // User found so get password and the user's record
        User_Record uRecord   DataBase.GetUserRecord( record ) ,
        String Password   DataIn.readUTF() ;

        // Check password
        if ( uRecord.Is_Password_OK ( Password ) )
        {
            // Good Password
            uRecord.Login_User () ,
            DataOut.writeBoolean ( true ) ,
            System.out.println("User "+ User_Name+" Logged In") ;
            return UNS_API.Local_User_Pass ,
        }
        else
        {
            // Bad Password
            DataOut.writeBoolean ( false ) ;
            return UNS_API.Local_User_Fail ,
        }
    }
}

//////////////////////////////////////////////
```

```
//  Logout_User
//
//  Arguments:
//      User_Name -> name of the user attempting to logout
//
//  Attempts to logout a user,
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
private int
Logout_User ( String User_Name )throws IOException
{
    // not much checking built in yet
    Close_Connection ( User_Name ) ;
    DataOut.writeBoolean ( true ) ;
    System.out.println("User "+ User_Name+ " Logged Out") ;
    return UNS_API.Local_User_Fail ,
}

////////////////////////////////////////////
//  Get_Heirarchy
//
//  Arguments:
//      User_Name -> name of the user attempting to get
//          its heirarchy
//      Type -> File Ssyetm or Application
//
//  Attempts to send the appropriate service key list to the
//  client
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
private int
Get_Heirarchy ( String User_Name, int Type ) throws IOException
{
    // Find user in database
    int record   DataBase.GetUserIndex( User_Name ) ,

    if ( record == -1 )
    {
        // User Not Found
        return UNS_API.Local_User_Fail ;
    }
    else
    {
        // User found
        if ( Type == Application )
        {
            // Wants Application Key list
            // Attempt to retreive and send to client
            User_Record uRecord = DataBase.GetUserRecord( record ) ;
            Heirarchy_Manager nHeir = uRecord.Get_Application_Heir () ,
            if ( nHeir != null )
            {
                nHeir.WriteHeirarchy (DataOut) ;
            }
            else
            {
                return UNS_API.Local_User_Fail ;
            }
        }
        else
        {
            // Wants File System Key list
            // Attempt to retreive and send to client
            User_Record uRecord = DataBase.GetUserRecord( record ) ;
            Heirarchy_Manager nHeir   uRecord.Get_File_System_Heir () ;
            if ( nHeir != null )
```

```
            {
                nHeir.WriteHeirarchy (DataOut) ,
            }
            else
            {
                return UNS_API.Local_User_Fail ,
            }
        }
    }
    return UNS_API.Local_User_Pass ;
}

////////////////////////////////////////////
//  Set_Heirarchy
//
//  Arguments:
//      User_Name -> name of the user attempting to set
//          its heirarchy
//      Type -> File Ssyetm or Application
//
//  Attempts to receive and store the appropriate service
//  key list from the client
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
private int
Set_Heirarchy ( String User_Name, int Type ) throws IOException
{
    // Attemt to find the user in the database
    int record   DataBase.GetUserIndex( User_Name ) ,
    if ( record == -1 )
    {
        // User Not Found
        return UNS_API.Local_User_Fail ,
    }
    else
    {
        // User found
        // So download key list and set appropriatly
        Heirarchy_Manager nHeir   new Heirarchy_Manager ( DataIn ) ;
        User_Record uRecord   DataBase.GetUserRecord( record ) ,
        if ( Type == Application )
        {
            uRecord.Set_Application_Heir (nHeir) ;
        }
        else
        {
            uRecord.Set_File_System_Heir (nHeir) ;
        }
    }
    return UNS_API.Local_User_Pass ,
}

////////////////////////////////////////////
//  Get_Config
//
//  Arguments:
//      User_Name -> name of the user attempting to get
//          its config
//
//  Attempts to send user configuration to the client
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
private int
Get_Config ( String User_Name ) throws IOException
{
    // Attemt to find the user in the database
```

```
    int record = DataBase.GetUserIndex( User_Name ) ,

    if ( record == -1 )
    {
        // User Not Found
        return UNS_API.Local_User_Fail ,
    }
    else
    {
        // User found
        // so get config and send
        User_Record uRecord   DataBase.GetUserRecord( record ) ,
        Config_Manager nConfig   uRecord.Get_User_Configuration () ,
        if ( nConfig != null )
        {
            nConfig.WriteStream (DataOut) ;
        }
        else
        {
            return UNS_API.Local_User_Fail ,
        }
    }
    return UNS_API.Local_User_Pass ,
}

////////////////////////////////////////////
//  Set_Config
//
//  Arguments:
//      User_Name -> name of the user attempting to set
//          its config
//
//  Attempts to send user configuration to the client
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
private int
Set_Config ( String User_Name ) throws IOException
{
    // Attempt to locate user
    int record   DataBase.GetUserIndex( User_Name ) ,
    if ( record == -1 )
    {
        // User Not Found
        return UNS_API.Local_User_Fail ,
    }
    else
    {
        // User found
        User_Record uRecord   DataBase.GetUserRecord( record ) ;
        Config_Manager nConfig = new Config_Manager ( DataIn ) ;
        uRecord.Set_User_Configuration ( nConfig ) ,
    }
    return UNS_API.Local_User_Pass ,
}

////////////////////////////////////////////
//  Verify_Key
//
//  Arguments:
//      none
//
//  Attempts to verify that the sent key is valid and
//  belongs to a logged in user
//
//  throws IOException if any port errors occur for the calling
//  thread to handle
private int
```

```
Verify_Key ( ) throws IOException
{
    // Get the servcie key
    byte nBuffer [] = new byte [ DataIn.readInt () ] ,
    DataIn.read( nBuffer, 0, nBuffer.length ) ,
    Service_Key nKey   new Service_Key ( nBuffer ) ;

    // Attempt to locate the user
    int record = DataBase.GetUserIndex( nKey.getUserName () ) ,
    if ( record == -1 )
    {
        // User Not Found
        DataOut.writeBoolean( false ) ,
        return UNS_API.Local_User_Fail ,
    }
    else
    {
        // User found

        // So get user's record
        User_Record uRecord   DataBase.GetUserRecord( record ) ;

        // Get applkication key list and try to locate key
        Heirarchy_Manager nHeir   uRecord.Get_Application_Heir () ;
        int index = nHeir.GetKeyIndex ( nKey ) ,

        // Key found?
        if ( index < 0 )
        {
            // Not found
            // Get File system keys and check
            nHeir   uRecord.Get_File_System_Heir () ,
            index   nHeir.GetKeyIndex ( nKey ) ;

            // Key found ?
            if ( index < 0 )
            {
                // Key not found
                DataOut.writeBoolean( false ) ;
                return UNS_API.Local_User_Pass ;
            }
        }
        // Key found
        DataOut.writeBoolean( true ) ,
        return UNS_API.Local_User_Pass ,
    }
}

////////////////////////////////////////////
//  Close_Connection
//
//  Arguments:
//      User_Name -> name of the user attempting to
//          close it's connection
//
//  Attempts to log out the user and close the connection
public void
Close_Connection ( String User_Name )
{
    // Find user in database
    int record = DataBase.GetUserIndex( User_Name ) ,
    if ( record != -1 )
    {
        // User found so logout
        User_Record uRecord   DataBase.GetUserRecord( record ) ;
        uRecord.Logout_User () ;
    }
}
```

```
}
```

BootStrap

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    Bootstrap loader
*
*    Jeff Harman
*    Date        Revision    Comment
*    4/1/97      1.0         Creation
*
*   Intended to eventually load the information from file based resource(s)
*   to fill in the user database.  Curently load static data that is defined
*   by member functions.
*/

/**
*   Package: UNSServer
*       Package intended to be repository of all UNSServer objects
*/
package UNSServer ,

/**
*   System Imports
*/
import java.lang.* ,
import java.net.* ,
import java.io.* ,
import java.awt.* ;
import java.util.* ,
/**
*   Custom Imports
*/
import Keys.* ,
import UNSUtils.* ,

/**
*   Class Definition
*/
public
class BootStrap
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  Host -> The name of the machine that this server resides on
    private
    String Host = null ;
    // Data_File -> Data file to load info from (not used yet)
    private static final
    String Data_File = "UserData.txt" ,
    // UserNames -> Data file to load user names from
```

```
private static final
String UserNames = "UNSUsers.dat" ;
// FileHosts -> Data file to load FSS Host info from
private static final
String FileHosts   "FSSHosts.dat" ,
// NameList -> User Name List (from file)
private
Vector NameList = null ,
// NameList -> User Password List (from file)
private
Vector PassList   null ,
// HostList -> FSS Host Name List (from file)
private
Vector HostList   null ,
// PathList -> FSS Host User Path List (from file)
private
Vector PathList   null ;
// HostName -> FSS Host Machione Name (cities) (from file)
private
Vector HostName   null ;

//////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      HostName -> The name of the machine that this server resides on
//
//  Sets the host name for UNS Server User Information generation
//  Loads data files and sets up stored info (name list etc. )
//
public
BootStrap ( String ThisHostName )
{
    // Set the local host to the supplied host
    Host   new String ( ThisHostName ) ;

    NameList = new Vector ( 0 ) ,
    PassList = new Vector ( 0 ) ;
    HostList = new Vector ( 0 ) ,
    PathList   new Vector ( 0 ) ,
    HostName   new Vector ( 0 ) ;

    try
    {
        // Load Information about Users
        FileInputStream ServerFile   new FileInputStream ( UserNames ) ,
        DataInputStream DataIn   new DataInputStream ( ServerFile ) ,

        while ( ServerFile.available() != 0 )
        {
            String Input1   DataIn.readLine () ,
            String Input2 = DataIn.readLine () ;

            NameList.addElement ( Input1 ) ,
            PassList.addElement ( Input2 ) ,
        }

        // Load Information about FSS Hosts
        ServerFile = new FileInputStream ( FileHosts ) ,
        DataIn = new DataInputStream ( ServerFile ) ,

        while ( ServerFile.available() != 0 )
        {
            String Input0   DataIn.readLine () ;
            String Input1   DataIn.readLine () ;
            String Input2 = DataIn.readLine () ;

            HostName.addElement ( Input0 ) ,
```

```
            HostList.addElement ( Input1 ) ,
            PathList.addElement ( Input2 ) ,
        }

        // Debug Output
        //for ( int i   0 , i < HostName.size () ; i++ )
        //{
        //    System.out.println ( (String)HostName.elementAt( i ) ) ,
        //}
    }
    catch ( FileNotFoundException e )
    {
        System.exit(1) ;
        // Error loading srartup info, UNS Server will not
        //  function without them so shut down.
    }
    catch ( IOException e )
    {
        System.exit(1) ,
        // Error loading srartup info, UNS Server will not
        //  function without them so shut down.
    }

}

////////////////////////////////////////////
//  Create_New_Database
//
//  Arguments:
//      none
//
//  builds a new user database with three users
//  (static generation) no varaibility
//
//  Returns the new User_Database
//
public User_Database
Create_New_Database ()
{
    // Create the blank Database
    User_Database New_Data   new User_Database () ,

    for ( int i = 0 ; i < NameList.size () ; i++ )
    {
        // Add User Sam_Snead
        New_Data.AddUser ( List_User_Add ( i ) ) ,
    }

    return New_Data ,
}

////////////////////////////////////////////
//  List_User_Add
//
//  Arguments:
//      RecordNumber -> The entry in the list of users to add
//
//  Fills in all the information for the user and creates a
//  set of service keys for the user to use
private User_Record
List_User_Add ( int RecordNumber )
{
    String User_name = (String)NameList.elementAt ( RecordNumber ) ,
    String User_Pass = (String)PassList.elementAt ( RecordNumber ) ,

    User_Record New_User = new User_Record () ;
    User_Info_Manager New_uData   new User_Info_Manager () ;
```

```
New_uData.SetName ( User_name ) ;
New_uData.SetPassword ( User_Pass ) ,

New_User.Set_User_Information ( New_uData ) ;

Heirarchy_Manager HMan   new Heirarchy_Manager () ;

// Service key to go in the application list
Service_Key SKey   new Service_Key () ;
SKey.setUserName ( User_name+"@"+Host ) ;
SKey.setUserHost ( Host ) ;
SKey.setServiceRequest ( "Application 1" ) ,
SKey.setAdditionalInfo ( "Application 1 Display Information" ) ;
byte Test []   { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 } ;
SKey.setKey ( Test  ) ;
SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

// Add the filled in key
HMan.AddKey ( SKey ) ,

// Service key to go in the application list
SKey   new Service_Key () ,
SKey.setUserName ( User_name+"@"+Host ) ;
SKey.setUserHost ( Host ) ;
SKey.setServiceRequest ( "Application 2" ) ,
SKey.setAdditionalInfo ( "Application 2 Display Information" ) ,
SKey.setKey ( Test  ) ;
SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

HMan.AddKey ( SKey ) ,

SKey = new Service_Key () ;
SKey.setUserName ( User_name+"@"+Host ) ,
SKey.setUserHost ( Host ) ;
SKey.setServiceRequest ( "Application 3" ) ;
SKey.setAdditionalInfo ( "Application 3 Display Information" ) ;
SKey.setKey ( Test  ) ,
SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

HMan.AddKey ( SKey ) ,

New_User.Set_Application_Heir ( HMan ) ;

HMan   new Heirarchy_Manager () ,

for ( int i = 0 , i < HostName.size () ; i++ )
{
    SKey   new Service_Key () ,
    SKey.setUserName ( User_name ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( (String)HostName.elementAt(i) ) ;
    SKey.setAdditionalInfo ( (String)HostName.elementAt(i)+" Files" ) ,
    SKey.setKey ( SetBytefield2 ( i )  ) ;
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

    // Add the filled in key
    HMan.AddKey ( SKey ) ;
}

// Add the filled heirarchy to the user record
New_User.Set_File_System_Heir ( HMan ) ;

Config_Manager New_Config = new Config_Manager () ;

Color New_Color = new Color ( 10, 20, 30 ) ,

New_Config.SetForegroundColor ( New_Color ) ;
```

```
    New_User.Set_User_Configuration ( New_Config ) ,

    return New_User ;
}

////////////////////////////////////////////
//  SetBytefield2
//
//  Arguments:
//      RecordNumber -> The entry in the list of FSS Hosts to add
//
//  Fills in all the information for the FSS Host and creates a
//  bytefield for a FSS Service key with the appropriate info
private byte []
SetBytefield2 ( int RecordNumber )
{
    String Host    (String)HostList.elementAt ( RecordNumber ) ,
    String BasePath    (String)PathList.elementAt ( RecordNumber ) ,
    int    FSSPort = 3011 ,

    ByteArrayOutputStream Byter   new ByteArrayOutputStream () ;
    DataOutputStream IODev = new DataOutputStream ( Byter ) ;

    try
    {
        IODev.writeUTF ( Host ) ;
        IODev.writeInt ( FSSPort ) ,
        IODev.writeUTF ( BasePath ) ,
    }
    catch ( IOException e )
    {
        return null ;
    }

    return Byter.toByteArray() ;
}


////////////////////////////////////////////
//  Create_New_Database2
//
//  Arguments:
//      none
//
//  builds a new user database with three users
//  (static generation) no varaibility
//
//  Returns the new User_Database
//
public User_Database
Create_New_Database2 ()
{
    // Create the blank Database
    User_Database New_Data = new User_Database () ;

    // Add User Sam_Snead
    New_Data.AddUser ( Add_Sam_Snead () ) ;

    // Add User John_Doe
    New_Data.AddUser ( Add_John_Doe () ) ,

    // Add User Jane_Smith
    New_Data.AddUser ( Add_Generic ( "Jane_Smith" ) ) ;

    return New_Data ;
}

////////////////////////////////////////////
```

```
//  Add_John_Doe
//
//  Arguments:
//      none
//
//  Fills in all the information for the user and creates a
//  set of service keys for the user to use
private User_Record
Add_John_Doe ()
{
    // Create the record to be filled in
    User_Record New_User = new User_Record () ,

    // New user information manager for user
    User_Info_Manager New_uData   new User_Info_Manager () ,
    New_uData.SetName ( "John_Doe" ) ;
    New_uData.SetPassword ( "Foobar" ) ;

    // Store user information manager in the user record
    New_User.Set_User_Information ( New_uData ) ;

    // Application service key heirarchy
    Heirarchy_Manager HMan   new Heirarchy_Manager () ;

    // Service key to go in the application list
    Service_Key SKey   new Service_Key () ;
    SKey.setUserName ( "John_Doe@"+Host ) ,
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "Application 1" ) ;
    SKey.setAdditionalInfo ( "Application 1 Display Information" ) ,
    byte Test []   { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 } ;
    SKey.setKey ( Test  ) ;
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

    // Add the filled in key
    HMan.AddKey ( SKey ) ,

    // Service key to go in the application list
    SKey = new Service_Key () ,
    SKey.setUserName ( "John_Doe@"+Host ) ,
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "Application 3" ) ,
    SKey.setAdditionalInfo ( "Application 3 Display Information" ) ,
    SKey.setKey ( Test  ) ;
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

    // Add the filled in key
    HMan.AddKey ( SKey ) ;

    // Add the filled heirarchy to the user record
    New_User.Set_Application_Heir ( HMan ) ,

    // File System service key heirarchy
    HMan = new Heirarchy_Manager () ,

    // Service key to go in the file system list
    SKey = new Service_Key () ,
    SKey.setUserName ( "John_Doe" ) ,
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "New York" ) ;
    SKey.setAdditionalInfo ( "New York Files" ) ,
    SKey.setKey ( SetBytefield( 0 )  ) ,
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

    // Add the filled in key
    HMan.AddKey ( SKey ) ;

    // Service key to go in the file system list
```

```
    SKey = new Service_Key () ;
    SKey.setUserName ( "John_Doe" ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "San Francisco" ) ;
    SKey.setAdditionalInfo ( "San Francisco Files" ) ,
    SKey.setKey ( SetBytefield( 1 )  ) ,
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

    // Add the filled in key
    HMan.AddKey ( SKey ) ;

    // Service key to go in the file system list
    SKey = new Service_Key () ;
    SKey.setUserName ( "John_Doe" ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "London" ) ,
    SKey.setAdditionalInfo ( "London Files" ) ,
    SKey.setKey ( SetBytefield( 3 )  ) ,
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

    // Add the filled in key
    HMan.AddKey ( SKey ) ;

    // Add the filled heirarchy to the user record
    New_User.Set_File_System_Heir ( HMan ) ;

    Config_Manager New_Config   new Config_Manager () ,

    Color New_Color   new Color ( 10, 20, 30 ) ,

    New_Config.SetForegroundColor ( New_Color ) ;

    New_User.Set_User_Configuration ( New_Config ) ,

    return New_User ,
}

//////////////////////////////////////////////
//  Add_Sam_Snead
//
//  Arguments:
//      none
//
//  Fills in all the information for the user and creates a
//  set of service keys for the user to use
private User_Record
Add_Sam_Snead ()
{
    // Create the record to be filled in
    User_Record New_User = new User_Record () ;

    // New user information manager for user
    User_Info_Manager New_uData   new User_Info_Manager () ;

    New_uData.SetName ( "Sam_Snead" ) ,
    New_uData.SetPassword ( "Quark" ) ;

    // Store user information manager in the user record
    New_User.Set_User_Information ( New_uData ) ,

    // Application service key heirarchy
    Heirarchy_Manager HMan   new Heirarchy_Manager () ;

    // Service key to go in the application list
    Service_Key SKey = new Service_Key () ;
    SKey.setUserName ( "Sam_Snead@"+Host ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "Application 1" ) ,
```

```
SKey.setAdditionalInfo ( "Application 1 Display Information" ) ;
byte Test [] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 } ;
SKey.setKey ( Test  ) ;
SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

// Add the filled in key
HMan.AddKey ( SKey ) ;

// Service key to go in the application list
SKey = new Service_Key () ;
SKey.setUserName ( "Sam_Snead@"+Host ) ,
SKey.setUserHost ( Host ) ;
SKey.setServiceRequest ( "Application 2" ) ;
SKey.setAdditionalInfo ( "Application 2 Display Information" ) ,
SKey.setKey ( Test  ) ,
SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

// Add the filled in key
HMan.AddKey ( SKey ) ,

// File System service key heirarchy
New_User.Set_Application_Heir ( HMan ) ;

HMan   new Heirarchy_Manager () ,

// Service key to go in the file system list
SKey   new Service_Key () ,
SKey.setUserName ( "Sam_Snead" ) ,
SKey.setUserHost ( Host ) ;
SKey.setServiceRequest ( "New York" ) ,
SKey.setAdditionalInfo ( "New York Files" ) ;
SKey.setKey ( SetBytefield( 0 )  ) ;
SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

// Add the filled in key
HMan.AddKey ( SKey ) ;

// Service key to go in the file system list
SKey   new Service_Key () ,
SKey.setUserName ( "Sam_Snead" ) ,
SKey.setUserHost ( Host ) ;
SKey.setServiceRequest ( "San Francisco" ) ,
SKey.setAdditionalInfo ( "San Francisco Files" ) ,
SKey.setKey ( SetBytefield( 1 )  ) ,
SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

// Add the filled in key
HMan.AddKey ( SKey ) ,

// Service key to go in the file system list
SKey   new Service_Key () ,
SKey.setUserName ( "Sam_Snead" ) ,
SKey.setUserHost ( Host ) ;
SKey.setServiceRequest ( "Hong Kong" ) ;
SKey.setAdditionalInfo ( "Hong Kong Files" ) ,
SKey.setKey ( SetBytefield( 2 )  ) ,
SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

// Add the filled in key
HMan.AddKey ( SKey ) ,

New_User.Set_File_System_Heir ( HMan ) ;

Config_Manager New_Config   new Config_Manager () ,

New_User.Set_User_Configuration ( New_Config ) ;

return New_User ,
```

```
}

private byte []
SetBytefield ( int CityChoice )
{
    String Host ,
    String BasePath ,
    int    FSSPort = 3011 ,

    switch ( CityChoice )
    {
        case 0 .
            Host   new String ( "hpcims17.isc.rit.edu" ) ,
            BasePath = new String ( "/home/stu5/kam3011/home17" ) ;
            break ;
        case 1 :
            Host = new String ( "hpcims18.isc.rit.edu" ) ,
            BasePath = new String ( "/home/stu5/kam3011/home18" ) ;
            break ,
        case 2 .
            Host   new String ( "hpcims19.isc.rit.edu" ) ;
            BasePath   new String ( "/home/stu5/kam3011/home19" ) ,
            break ,
        default
            Host   new String ( "hpcims20.isc.rit.edu" ) ,
            BasePath   new String ( "/home/stu5/kam3011/home20" ) ,
            break ;
    }

    ByteArrayOutputStream Byter   new ByteArrayOutputStream () ;
    DataOutputStream IODev   new DataOutputStream ( Byter ) ,

    try
    {
        IODev.writeUTF ( Host ) ,
        IODev.writeInt ( FSSPort ) ,
        IODev.writeUTF ( BasePath ) ,
    }
    catch ( IOException e )
    {
        return null ,
    }

    return Byter.toByteArray() ;
}

////////////////////////////////////////////
//  Add_Generic
//
//  Arguments:
//      User_name -> The name of the user that this record is for
//
//  Fills in all the information for the user and creates a
//  set of service keys for the user to use
private User_Record
Add_Generic ( String User_name )
{
    User_Record New_User   new User_Record () ,
    User_Info_Manager New_uData   new User_Info_Manager () ,

    New_uData.SetName ( User_name ) ,
    New_uData.SetPassword ( "Passwd" ) ,

    New_User.Set_User_Information ( New_uData ) ;

    Heirarchy_Manager HMan = new Heirarchy_Manager () ;
```

```
    Service_Key SKey = new Service_Key () ;
    SKey.setUserName ( User_name+"@"+Host ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "Application 2" ) ;
    SKey.setAdditionalInfo ( "Application 2 Display Information" ) ;
    byte Test [] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 } ;
    SKey.setKey ( Test  ) ;
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

    HMan.AddKey ( SKey ) ;

    SKey   new Service_Key () ,
    SKey.setUserName ( User_name+"@"+Host ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "Application 3" ) ,
    SKey.setAdditionalInfo ( "Application 3 Display Information" ) ,
    SKey.setKey ( Test  ) ;
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

    HMan.AddKey ( SKey ) ;

    New_User.Set_Application_Heir ( HMan ) ;

    HMan   new Heirarchy_Manager () ,

    SKey   new Service_Key () ;
    SKey.setUserName ( User_name ) ,
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "San Francisco" ) ,
    SKey.setAdditionalInfo ( "San Francisco Files" ) ;
    SKey.setKey ( SetBytefield( 1 )  ) ;
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

    HMan.AddKey ( SKey ) ;

    SKey = new Service_Key () ,
    SKey.setUserName ( User_name ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "Hong Kong" ) ,
    SKey.setAdditionalInfo ( "Hong Kong Files" ) ;
    SKey.setKey ( SetBytefield( 2 )  ) ,
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ,

    HMan.AddKey ( SKey ) ;

    SKey = new Service_Key () ,
    SKey.setUserName ( User_name ) ;
    SKey.setUserHost ( Host ) ;
    SKey.setServiceRequest ( "London" ) ,
    SKey.setAdditionalInfo ( "London Files" ) ,
    SKey.setKey ( SetBytefield( 3 )  ) ;
    SKey.setHostPort ( UNS_API.Well_Known_Port ) ;

    HMan.AddKey ( SKey ) ,

    New_User.Set_File_System_Heir ( HMan ) ,

    Config_Manager New_Config = new Config_Manager () ,

    Color New_Color = new Color ( 50, 50, 50 ) ;

    New_Config.SetForegroundColor ( New_Color ) ,

    New_User.Set_User_Configuration ( New_Config ) ;

    return New_User ,
}
```

```
    ////////////////////////////////////////////////////
    //Public Member Variables
    //
}
```

Relay_Thread

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    Relay Thread Object
*
*    Jeff Harman
*    Date         Revision    Comment
*    3/26/97      1.0         Creation
*
*   Thread that handles data relay operations for proxy operation of
*   the UNS Server
*/

/**
*   Package: UNSServer
*       Package intended to be repository of all UNSServer objects
*/
package UNSServer ;

/**
*   System Imports
*/
import java.lang.* ,
import java.net.* ;
import java.io.* ,

/**
*   Class Definition
*/
public
class Relay_Thread extends Thread
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  DataIn -> port reading data from source
    private
    DataInputStream DataIn   null ;
    //  DataOut -> port writing data to source
    private
    DataOutputStream DataOut   null ,

    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
    //      InSocket -> port that data is coming in from
    //      OutSocket -> port that data is going out to
    //
    //  Sets up the ports for relaying data to and from
    public
    Relay_Thread ( Socket InSocket, Socket OutSocket )
    {
```

```
        try
        {
            // Create raw data ports
            InputStream rawDataIn   InSocket.getInputStream () ;
            OutputStream rawDataOut - OutSocket.getOutputStream () ;

            // Create structured data ports
            DataIn   new DataInputStream(rawDataIn) ;
            DataOut = new DataOutputStream(rawDataOut) ;
        }
        catch ( IOException f )
        {
        }
    }

    ////////////////////////////////////////////
    //  run
    //
    //  Arguments:
    //      none
    //
    //  attempts to begin the relay operation from input and output.
    //
    //  Thread will die if either the input or the output ports
    //  throws and IOException
    public
    void run ()
    {
        try
        {
            // Infine loop
            while ( true )
            {
                // Read then write
                DataOut.writeByte ( DataIn.readByte () ) ;
            }
        }
        catch ( IOException f )
        {
            // Kill Ports and exit
            DataOut   null ,
            DataIn   null ;
        }
    }
}
```

UNSClient

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    UNSClient
*
*    Jeff Harman
*    Date        Revision    Comment
*    4/1/97      1.0         Creation
*
*    Manage all requests for a client.  Is maintained only for the duration
*    of the clients life.
*/

/**
*   Package: UNSServer
*       Package intended to be repository of all UNSServer objects
*/
package UNSServer ;

/**
*   System Imports
*/
import java.lang.* ,
import java.net.* ,
import java.io.* ;
/**
*   Custom Imports
*/
import Keys.* ;
import UNSUtils.* ;

/**
*   Class Definition
*/
public
class UNSClient extends Thread
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  ClientConnection -> The socket that connects to the client
    private
    Socket ClientConnection   null ;
    //  rawDataIn -> unformatted port reading data from source
    private
    InputStream rawDataIn   null ;
    //  rawDataOut -> unformatted port writing data to source
    private
    OutputStream rawDataOut   null ;
    //  DataIn -> port reading data from source
```

```
private
DataInputStream DataIn = null ;
//  DataOut -> port writing data to source
private
DataOutputStream DataOut = null ,
//  DataMediator -> mediator to use for filling client requests
private
Mediator DataMediator   null ;
//  OutSocket -> The socket that connects to remote server
private
Socket OutSocket   null ;
// User_Name -> Name of the user of this thread
private
String User_Name   null ,
// Hset_Host -> Byte array that contains the DNS name of this server
private
byte User_Host []   null ;

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      Connection -> Socket that connects to the host
//      dataMediator -> Data aguisition source
//      inHost -> the DNS byte name of this host
//
//  Attempts to configure this thread to service client requests
//
public
UNSClient ( Socket Connection, Mediator dataMediator, byte inHost [] )
{
    try
    {
        DataMediator   dataMediator ;
        User_Host = inHost ,

        ClientConnection   Connection ;

        rawDataIn = Connection.getInputStream () ;
        rawDataOut   Connection.getOutputStream () ;

        DataIn   new DataInputStream(rawDataIn) ;
        DataOut   new DataOutputStream(rawDataOut) ,
    }
    catch ( IOException e )
    {
        System.out.println ("IO Error.\n") ,
    }
}

////////////////////////////////////////////
//  run
//
//  Arguments:
//      none
//
//  Handles all requests form a single client for the duration of
//  login (or attempt) or key validation request
//
public
void run ()
{
    try
    {
        // Preset the value of the loop controller
        int LoginValue   UNS_API.Local_User_Pass ,

        // Get the first request
```

```
            int PacketRequest = DataIn.readInt () ;

            if ( PacketRequest == UNS_API.Login )
            {
                // first request login
                // Get user name
                User_Name   new String ( DataIn.readUTF () ) ,

                // echo user name to screen
                System.out.println ( "User: "+User_Name ) ,
                // Extract the user host from the login name
                String sUserHost   UserHostName ( User_Name ) ,

                // Get the address of the client
                InetAddress Test   InetAddress.getByName ( sUserHost ) ;
                byte UserHost []   Test.getAddress () ;

                // Test for local client
                boolean Comapre   ( User_Host[0] == UserHost[0] ) &&
                                  ( User_Host[1] == UserHost[1] ) &&
                                  ( User_Host[2] == UserHost[2] ) &&
                                  ( User_Host[3] == UserHost[3] ) ,

                if ( Comapre )
                {
                    // Local User
                    System.out.println ( "Local Host User." ) ;
                    User_Name   new String ( UserNameOnly ( User_Name ) ) ,

                    // Attempt to login user
                    LoginValue   DataMediator.New_Request ( PacketRequest, ClientConnection, User_Name
) ;
                }
                else
                {
                    // Remote user
                    System.out.println ( "Remote Host User." ) ;
                    LoginValue   UNS_API.Non_Local_User ,
                }

                while (LoginValue == UNS_API.Local_User_Pass )
                {
                    // Handle all subsequent requests with the Mediator
                    PacketRequest   DataIn.readInt () ;
                    if ( PacketRequest == UNS_API.Login )
                    {
                        User_Name   new String ( DataIn.readUTF () ) ;
                        LoginValue   DataMediator.New_Request ( PacketRequest, ClientConnection,
User_Name ) ,
                    }
                    else
                    {
                        LoginValue   DataMediator.New_Request ( PacketRequest, ClientConnection,
User_Name ) ;
                    }
                }

                if ( LoginValue == UNS_API.Non_Local_User )
                {
                    // Proxy server, relay request to new server
                    OutSocket   new Socket ( sUserHost, UNS_API.Well_Known_Port ) ,
                    rawDataOut = OutSocket.getOutputStream () ,
                    DataOut   new DataOutputStream(rawDataOut) ;

                    // Re-request service with new server
                    DataOut.writeInt ( UNS_API.Login ) ;

                    DataOut.writeUTF ( User_Name ) ,
```

```
            System.out.println( "Relay Client" ) ;

            // create realy threads to handle connection
            Relay_Thread ClientThread   new Relay_Thread ( OutSocket, ClientConnection)  ,
            Thread CThread   new Thread ( ClientThread ) ,
            CThread.start () ;
            ClientThread   new Relay_Thread ( ClientConnection, OutSocket )  ,
            CThread = new Thread ( ClientThread ) ;
            CThread.start () ,

            // eliminate local connections to porst and
            // allow thread to terminate
            ClientConnection   null ,
            OutSocket   null ,
            DataIn   null ;
            DataOut = null ;
            rawDataIn   null ,
            rawDataOut = null ,
            CThread = null ;
            ClientThread   null ,
        }
    }
    else if ( PacketRequest == UNS_API.Verify_Key )
    {
        // First Request Verify Key
        DataMediator.New_Request ( PacketRequest, ClientConnection, null ) ,
    }
    // else
    // invalid first request (allow thread to terminate)

    System.out.println("Client Closing") ,
}
catch ( UnknownHostException e )
{
    // User specified host is invalid
    System.out.println("Client Closing") ,

    try
    {
        // Let client know that attempt to log in failed
        DataOut.writeBoolean ( false ) ,
    }
    catch ( IOException f )
    {
        // Port errors
        if ( User_Name != null )
        {
            DataMediator.Close_Connection ( User_Name ) ;
        }

        User_Name   null ;

        ClientConnection   null ,
        OutSocket = null ;
        DataIn = null ;
        DataOut   null ;
        rawDataIn   null ;
        rawDataOut   null ,

        stop () ,
    }
}
catch ( IOException e )
{
    // Port Errors
    System.out.println("Client Closing") ,
```

```
            if ( User_Name != null )
            {
                DataMediator.Close_Connection ( User_Name ) ,
            }

            User_Name   null ,

            ClientConnection = null ;
            DataIn   null ,
            DataOut = null ;
            rawDataIn = null ;
            rawDataOut = null ;
            OutSocket   null ,

            stop () ;
        }
    }

    ////////////////////////////////////////////
    //  UserHostName
    //
    //  Arguments:
    //      UserName -> full user name (including host)
    //
    //  assumes that an '@' is present in string
    //
    //  extracts user host name from login user name
    private String
    UserHostName ( String UserName )
    {
        int Position = User_Name.indexOf( '@' ) ;
        String Result - new String ( User_Name.substring(Position+1) ) ,
        return Result ;
    }

    ////////////////////////////////////////////
    //  UserNameOnly
    //
    //  Arguments:
    //      UserName -> full user name (including host)
    //
    //  assumes that an '@' is present in string
    //
    //  extracts user name from login user name
    private String
    UserNameOnly ( String UserName )
    {
        int Position   User_Name.indexOf( '@' ) ;
        String Result   new String ( User_Name.substring(0,Position) ) ,
        return Result ,
    }
}
```

UNSCommManager

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    UNS Communication Manager
*
*
*    Jeff Harman
*    Date        Revision    Comment
*    4/1/97      1.0         Creation
*
*   Constructor for UNSComMananger is all that is necessary
*   to spawn the UNS Server
*
*   Intended to accept new connections for UNS Server and spawn client
*   threads to handle them (keeping the original port open)
*/

/**
*   Package: UNSServer
*       Package intended to be repository of all UNSServer objects
*/
package UNSServer ;

/**
*   System Imports
*/
import java.io.* ,
import java.net.* ;
/**
*   Custom Imports
*/
import Keys.* ,
import UNSServer.* ;
import UNSUtils.* ,

/**
*   Class Definition
*/
public
class UNSCommManager
{
    ////////////////////////////////////////////////////////
    //Private Member Variables
    //
    // Mediator -> Communication Structure
    private
    Mediator dataMediator   null ;
    // DataBase -> User requests (Data Base Manager)
    private
    User_Database DataBase = null ;
```

```
// Local_Host -> Name of this machine
private String
Local_Host   null ;
// LHostAddrs -> Name of this machine in byte format
private byte LHostAddrs []   null ;

//////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      none
//
//  Creates all necessary data management objects to run the server
//  Creates and infinte loop to block on waiting for new client
//          connections.  Spawns a new client to handle each new
//          client connection.
//
public
UNSCommManager ()
{
    try
    {
        // Create sockets for communication
        Socket ReplySocket   null ;
        ServerSocket Key_Server = new ServerSocket ( UNS_API.Well_Known_Port ) ,

        // Determine the local host name (???.???.???.???)
        InetAddress LHost   InetAddress.getLocalHost () ,
        Local_Host   LHost.toString () ;
        LHostAddrs   LHost.getAddress () ;
        int Position = Local_Host.indexOf( '/' ) ;
        Local_Host   new String ( Local_Host.substring(Position+1) ) ,

        // Display host name and echo screen
        System.out.println("UNS Server:") ,
        System.out.println("-------------------------------") ;
        System.out.println( "Machine Name: "+Local_Host ) ,
        System.out.println( "-------------------------------") ,

        // Create Data Menebers
        BootStrap testLoad = new BootStrap (Local_Host) ,
        DataBase   testLoad.Create_New_Database () ,
        dataMediator   new Mediator ( DataBase ) ;

        // Loop and wait for new connections
        while ( true )
        {
            ReplySocket   Key_Server.accept () ,
            NewService ( ReplySocket) ,
        }
    }
    catch ( UnknownHostException e )
    {
        System.out.println ("IO Error.\n") ;
    }
    catch ( IOException e )
    {
        System.out.println ("IO Error.\n") ,
    }
}

//////////////////////////////////////////
//  NewService
//
//  Arguments:
//      cSocket -> Client connection socket
//
//  Spawn a thread to handle the new client
```

```
    private void
    NewService ( Socket cSocket )
    {
        System.out.println ("New Client") ;
        UNSClient ClientThread   new UNSClient ( cSocket, dataMediator, LHostAddrs ) ,
        // next two lines are probably not necessary
        // Thread CThread = new Thread ( ClientThread ) ,
        // CThread.start () ,

        // Start the thread
        ClientThread.start () ,
    }
}
```

UNSServer

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    UNSServer (& Main)
*
*    Spawn a new UNS Server
*/

/**
*   Custom Imports
*/
import UNSServer.* ,

/**
*   Class Definition
*/
public
class UNSServer
{

    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
    //      none
    //
    //  Spawn a new UNSCommManager
    //  (and thereby start a new UNS Server)
    public
    UNSServer ()
    {
        new UNSCommManager () ;
    }

    ////////////////////////////////////////////
    //  main
    //
    //  Arguments:
    //      args -> List of command line arguments (ignored)
    //
    //  Just create a new UNS Server
    //
    public static void
    main ( String args [] )
    {
        new UNSServer ( ) ,
    }
}
```

User_Database

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    User Database Object
*
*    Jeff Harman
*    Date         Revision    Comment
*    3/26/97      1.0         Creation
*
*   Designed to contain a list of user records
*   allows simple name serach and record retrevial
*   in addition to replacement, addition and deletion
*   of user records.
*/

/**
*   Package: UNSServer
*       Package intended to be repository of all UNSServer objects
*/
package UNSServer ;

/**
*   System Imports
*/
import java.lang.* ,
import java.io.* ;
import java.util.* ;
/**
*   Custom Imports
*/
import Keys.* ;
import UNSUtils.* ,

/**
*   Class Definition
*/
public
class User_Database
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  Record_List -> voector of all the user records
    private Vector
    Record_List   null ;

    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
```

```
//      none
//
//  creates an empty User_database
//
public
User_Database ()
{
    Record_List = new Vector () ,
}

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InStream -> Data that will constitute the user database
//
//  Builds a new database from the information
//    contained in a bytestream
//
public
User_Database ( byte InStream [] ) throws IOException
{
    // Create DataInput stream from the instream
    ByteArrayInputStream Byter   new ByteArrayInputStream (InStream) ;
    DataInputStream Input   new DataInputStream( Byter ) ;

    // Fill in the database from the stream
    StreamConstructor ( Input ) ;
}

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      Old_Database -> Set of information to be
//              duplicated here
//
//  Copy Constructor
//
public
User_Database ( User_Database Old_Database ) throws IOException
{
    // Get a strean from the old object
    ByteArrayInputStream Byter =
        new ByteArrayInputStream ( Old_Database.getDataStream () ) ,
    DataInputStream Input   new DataInputStream( Byter ) ,

    // And use it to construct the new object
    StreamConstructor ( Input ) ,
}

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InputStream -> Input stream that contains the information
//          needed to construct a new database object
//
//  Construct a new user database from the input stream
//
public
User_Database ( DataInputStream InputStream ) throws IOException
{
    // Call the stream constructor
    StreamConstructor ( InputStream ) ,
}

////////////////////////////////////////////
```

```
//  getDataStream
//
//  Arguments:
//      none
//
//  return a byte array that contains all the information
//  from this database
public byte []
getDataStream ()
{
    // Not implemented yet
    return new byte[256] ,
}

////////////////////////////////////////////
//  StreamConstructor
//
//  Arguments:
//      InputStream -> Stream that contains all the
//          information contained in a user database
//
// throws IOException if there is any error reading the data
//
//  Builds a new user database from the supplied stream
private void
StreamConstructor ( DataInputStream InputStream ) throws IOException
{
    // Get the number of records
    int User_Count   InputStream.readInt ( ) ,

    // Build a new vector to contain them
    Record_List   new Vector ( User_Count ) ,

    for ( int i   0 , i < User_Count ; i++ )
    {
        // Extract eaach record from the stream
        AddUser ( new User_Record ( InputStream ) ) ;
    }
}

////////////////////////////////////////////
//  AddUser
//
//  Arguments:
//      NewUser -> Record to be added to the database
//
//  Add a new user record to the database
//
public void
AddUser ( User_Record NewUser )
{
    Record_List.addElement(NewUser) ,
}

////////////////////////////////////////////
//  DeleteUser
//
//  Arguments:
//      Index -> Index of the record to be deleted from the database
//
//  delete a user record from the database at the specified index
//
public void
DeleteUser ( int Index )
{
    Record_List.removeElementAt( Index ) ,
}
```

```
    ////////////////////////////////////////////
    //  GetUserIndex
    //
    //  Arguments:
    //      User_Name -> User neame of the desired record
    //
    //  find a user record index by the user name
    //  returns -1 if failed
    //
    public int
    GetUserIndex ( String User_Name )
    {
        // Search the list until found
        for ( int i   0 ; i < Record_List.size() , i++ )
        {
            User_Record Next_Record   (User_Record) Record_List.elementAt ( i ) ,

            if ( Next_Record.Is_Record_User ( User_Name ) )
                return i ,
        }

        // Else return not found
        return -1 ;
    }

    ////////////////////////////////////////////
    //  GetUserRecord
    //
    //  Arguments:
    //      Index -> index of the desired record
    //
    //  get a user record by it's index
    //
    public User_Record
    GetUserRecord ( int Index )
    {
        return (User_Record) Record_List.elementAt ( Index ) ,
    }
}
```

User_Record

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    User Record Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    3/26/97     1.0         Creation
*
*   Manages the information for a single user.
*   Intended to be used with the User_Database
*
*/

/**
*   Package: UNSServer
*       Package intended to be repository of all UNSServer objects
*/
package UNSServer ,

/**
*   System Imports
*/
import java.lang.* ;
import java.net.* ,
import java.io.* ,
/**
*   Custom Imports
*/
import Keys.* ;
import UNSUtils.* ;

/**
*   Class Definition
*/
class User_Record
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  Application_List -> List of service keys for Applications
    private Heirarchy_Manager
    Application_List = null ;
    //  File_System_List -> List of service keys for the File System
    private Heirarchy_Manager
    File_System_List = null ;
    // User_Information -> User Name and Password for this record
    private User_Info_Manager
    User_Information = null ;
    // User_Config -> User system configuration records
```

```
private Config_Manager
User_Config = null ;
// User_Logged_In -> Is this user logged in?
private boolean
User_Logged_In = false ,

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      none
//
//  Create a blank user record
//
public
User_Record ()
{
    Application_List = new Heirarchy_Manager () ;
    File_System_List = new Heirarchy_Manager () ;
    User_Information = new User_Info_Manager () ;
    User_Config   new Config_Manager () ;
    User_Logged_In   false ;
}

////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      InputStream -> Data Stream that contains information
//          needed to construct a User_record.
//
//  Create a user record from the informayion contained
//  in the InputStream.
//
public
User_Record ( DataInputStream InputStream ) throws IOException
{
    // Fill in each of the data objects from the data stream
    Application_List   new Heirarchy_Manager (InputStream) ;
    File_System_List   new Heirarchy_Manager (InputStream) ,
    User_Information   new User_Info_Manager (InputStream) ,
    User_Config   new Config_Manager ( InputStream ) ,
    User_Logged_In   false ;
}

////////////////////////////////////////////
//  WriteUserRecord
//
//  Arguments:
//      OutputStream -> Data Stream that will contain information
//          needed to construct a User_record.
//
//  Output a user record to the indicated Output Stream.
//
public void
WriteUserRecord ( DataOutputStream OutputStream ) throws IOException
{
    // Write out each of the data objects to the data stream
    Application_List.WriteHeirarchy( OutputStream ) ,
    File_System_List.WriteHeirarchy( OutputStream ) ,
    User_Information.WriteStream ( OutputStream ) ,
    User_Config.WriteStream ( OutputStream ) ;
}

////////////////////////////////////////////
//  Is_Record_User
//
//  Arguments:
```

```
//      User_Name -> User name to be compared to the user
//          name in the record
//
//  Returns:
//      true -> if the input name and the recorded name are the same
//      false -> if the input name and the recorded name are NOT the same
//
public boolean
Is_Record_User ( String User_Name )
{
    return User_Information.CheckName ( User_Name ) ,
}

////////////////////////////////////////////
//  Is_Password_OK
//
//  Arguments:
//      User_Password -> User password to be compared to the user
//          password in the record
//
//  Returns:
//      true -> if the input password and the recorded password are the same
//      false -> if the input password and the recorded password are NOT the same
//
public boolean
Is_Password_OK ( String User_Password )
{
    return User_Information.CheckPassword ( User_Password ) ,
}

////////////////////////////////////////////
//  Login_User
//
//  Arguments:
//      none
//
//  Set the logged in status to true
//
public void
Login_User ()
{
    User_Logged_In   true ,
}

////////////////////////////////////////////
//  Logout_User
//
//  Arguments:
//      none
//
//  Set the logged in status to false
//
public void
Logout_User ()
{
    User_Logged_In   false ;
}

////////////////////////////////////////////
//  Is_User_Logged_In
//
//  Arguments:
//      none
//
//  return the status of the user logged in
//
public boolean
Is_User_Logged_In ()
```

```
{
    return User_Logged_In ,
}

////////////////////////////////////////////
//  Get_Application_Heir
//
//  Arguments:
//      none
//
//  return the application heirarchy for the user
//
public Heirarchy_Manager
Get_Application_Heir ()
{
    try
    {
        return new Heirarchy_Manager( Application_List ) ;
    }
    catch ( IOException  e )
    {
        return null ,
    }
}

////////////////////////////////////////////
//  Get_File_System_Heir
//
//  Arguments:
//      none
//
//  return the file system heirarchy for the user
//
public Heirarchy_Manager
Get_File_System_Heir ()
{
    try
    {
        return new Heirarchy_Manager( File_System_List ) ,
    }
    catch ( IOException  e )
    {
        return null ;
    }
}

////////////////////////////////////////////
//  Get_User_Configuration
//
//  Arguments:
//      none
//
//  return the configuration for the user
//
public Config_Manager
Get_User_Configuration ()
{
    return new Config_Manager( User_Config ) ,
}

////////////////////////////////////////////
//  Set_Application_Heir
//
//  Arguments:
//      NewHeir -> Application Key list to be stored for
//          the user.
//
//  Replaces the old Application Heiracrhy for the user
```

```
    //  with the new one.
    //
    public void
    Set_Application_Heir ( Heirarchy_Manager NewHeir )
    {
        try
        {
            Application_List   new Heirarchy_Manager( NewHeir ) ,
        }
        catch ( IOException  e )
        {
        }
    }

    ////////////////////////////////////////////
    //  Set_File_System_Heir
    //
    //  Arguments:
    //      NewHeir -> File System Key list to be stored for
    //          the user.
    //
    //  Replaces the old File System Heiracrhy for the user
    //  with the new one.
    //
    public void
    Set_File_System_Heir ( Heirarchy_Manager NewHeir )
    {
        try
        {
            File_System_List   new Heirarchy_Manager( NewHeir ) ;
        }
        catch ( IOException  e )
        {
        }
    }

    ////////////////////////////////////////////
    //  Set_User_Configuration
    //
    //  Arguments:
    //      NewConfig -> configuration to be stored for
    //          the user.
    //
    //  Replaces the old configuration for the user
    //  with the new one.
    //
    public void
    Set_User_Configuration ( Config_Manager NewConfig )
    {
        User_Config   new Config_Manager( NewConfig ) ;
    }

    ////////////////////////////////////////////
    //  Set_User_Information
    //
    //  Arguments:
    //      NewUse -> New user name and password.
    //
    //  Replaces the old user name and password
    //  with the new ones.
    //
    public void
    Set_User_Information ( User_Info_Manager NewUse )
    {
        User_Information   new User_Info_Manager ( NewUse ) ,
    }
}
```

AS Server

File List:
ASClient.java
ASCommManager.java
ASServer.java

ASClient

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    ASClient Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    3/26/97     1.0         Creation
*
*   Individual thread that handles a user request for information
*   from an AS Server.
*
*   Two applictaion defined:
*       TextEditor -> One file, demonstration only
*               "Application 1"
*               "Application 3"
*       TextEditor -> Nine files, (See Keith Miller's Thesis)
*               "Application 2"
*/

/**
*   Package: ASServer
*       Package intended to be repository of all AS Server object
*/
package ASServer ,

/**
*   System Import(s)
*/
import java.lang.* ;
import java.net.* ,
import java.io.* ,
/**
*   Custom Import(s)
*/
import Keys.* ;

public
class ASClient extends Thread
{
    ////////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  ClientConnection -> The socket that connects to the client
    private
    Socket ClientConnection   null ;
    //  rawDataIn -> unformatted port reading data from source
    private
    InputStream rawDataIn   null ;
    //  rawDataOut -> unformatted port writing data to source
```

```
private
OutputStream rawDataOut = null ,
//  DataIn -> port reading data from source
private
DataInputStream DataIn   null ;
//  DataOut -> port writing data to source
private
DataOutputStream DataOut   null ,
//  Applictaion -> indicates which application is
//                 being requested by the client
private
int Application = 0 ;
//  CKey -> Channel key used to verify connection
private
Channel_Key CKey   null ;


////////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      Connection -> Socket that connects to the host
//
//  Attempts to configure this thread to service client requests
//
public
ASClient ( Socket Connection )
{
    try
    {
        ClientConnection   Connection ,

        rawDataIn   Connection.getInputStream () ;
        rawDataOut   Connection.getOutputStream () ;

        DataIn   new DataInputStream(rawDataIn) ;
        DataOut   new DataOutputStream(rawDataOut) ;
    }
    catch ( IOException e )
    {
        System.out.println ("IO Error.\n") ,
    }
}

////////////////////////////////////////////
//  run
//
//  Arguments:
//      none
//
//  Handles all requests form a single client for the duration of
//  connection (or attempt)
//
public
void run ()
{
    try
    {
        // Get the request
        int PacketRequest   DataIn.readInt () ,

        // Has to start with a new service request
        if ( PacketRequest == NEWSERVICE )
        {
            // Start New Service
            System.out.println("Starting New Service:") ,

            Start_Service () ,
```

```
            // Get next request
            PacketRequest   DataIn.readInt () ;

            // Has to get application next
            if ( PacketRequest == GETAPPLICATION )
            {
                // Send Application
                System.out.println("Sending Application:") ;

                Send_Application () ;

                // Get last request
                PacketRequest   DataIn.readInt () ;

                // has to be close service
                if ( PacketRequest == CLOSESERVICE )
                {
                    // Finished
                    System.out.println("Close of Service:") ,

                    Close_Service () ,
                }
            }
        }

        System.out.println("Client Closing") ;
    }
    catch ( IOException e )
    {
        System.out.println("Client Closing") ,

        ClientConnection = null ;
        DataIn   null ,
        DataOut   null ;
        rawDataIn   null ;
        rawDataOut   null ;

        stop () ;
    }
}

////////////////////////////////////////////
//  Start_Service
//
//  Arguments:
//      none
//
//  Handles the intital start of the connection
//  in response to a start new request.
//
//  Validates the service key and determines the
//  appropriate application that this client will
//  be requesting.
//
private void
Start_Service () throws IOException
{
    try
    {
        // Get service key buffer length
        int Length   DataIn.readInt ( ) ,

        // Read service key buffer
        byte Buffer []   new byte [ Length ] ;
        DataIn.readFully ( Buffer, 0, Length ) ;

        // Build a new service key from the data
```

```
        Service_Key SKey   new Service_Key ( Buffer ) ,

        // Extract the service request
        String Request = SKey.getServiceRequest () ,

        // Blank byte field to put in channel key
        byte bKey []   new byte [10] ;

        if ( Request.equals ( "Application 1" ) )
        {
            // Security Verification of key here
            Application   1 ,
            for ( int i   0 , i < 10 ; i++ )
                bKey[i]   1 ;
        }
        else if ( Request.equals ( "Application 2" ) )
        {
            // Security Verification of key here
            Application   2 ,
            for ( int i   0 ; i < 10 , i++ )
                bKey[i] = 2 ,
        }
        else if ( Request.equals ( "Application 3" ) )
        {
            // Security Verification of key here
            Application   3 ,
            for ( int i   0 ; i < 10 , i++ )
                bKey[i]   3 ,
        }
        else
        {
            // Bad request (no service will be performed)
            DataOut.writeBoolean ( false ) ,
            // Will cause connection to close
            throw new IOException () ,
        }

        // Build a new channel key for conversation
        CKey = new Channel_Key () ;
        CKey.setUserName( SKey.getUserName () ) ,
        CKey.setKey( bKey ) ,

        // Buld a transmission buffer out of the channel key
        byte CKeyBuf[]   CKey.getKeyStream () ,

        // Reply to client that everything is OK
        DataOut.writeBoolean ( true ) ,
        // and send the channel key out
        DataOut.writeInt ( CKeyBuf.length ) ;
        DataOut.write ( CKeyBuf, 0 , CKeyBuf.length ) ;
    }
    catch ( IOException e )
    {
        throw e ;
    }
}

////////////////////////////////////////////
//  Send_Application
//
//  Arguments:
//      none
//
//  Sends the requested application to the client
//
private void
Send_Application () throws IOException
{
```

```
    try
    {
        // Get the channel key to verify that client is
        // good
        int Length   DataIn.readInt ( ) ;
        byte CBuffer []   new byte [Length] ;
        DataIn.read ( CBuffer, 0, Length ) ;
        Channel_Key NewCKey   new Channel_Key ( CBuffer ) ,

        // Compare the key with the local reference
        if ( CKey.Compare ( NewCKey ) )
        {
            // Key OK so send the appropriate application
            byte ApBuffer [] ,
            if ( Application == 2)
            {
                ApBuffer   GetFileManager () ;
            }
            else
            {
                ApBuffer = GetApplication () ;
            }

            // Write the application to the port
            DataOut.writeBoolean ( true ) ;
            DataOut.writeInt ( ApBuffer.length ) ,
            DataOut.write ( ApBuffer, 0, ApBuffer.length ) ;
            DataOut.flush () ;
        }
        else
        {
            DataOut.writeBoolean ( false ) ,
            throw new IOException () ,
        }
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  Close_Service
//
//  Arguments:
//      none
//
//  Finishes the service and closes the connection
//
private void
Close_Service () throws IOException
{
    try
    {
        // Get the channel key to compare to the
        // local reference key
        int Length = DataIn.readInt ( ) ,
        byte CBuffer [] = new byte [Length] ,
        DataIn.read ( CBuffer, 0, Length ) ;
        Channel_Key NewCKey   new Channel_Key ( CBuffer ) ;

        // Compare the keys
        if ( CKey.Compare ( NewCKey ) )
        {
            // Keys match (normal shutdown)
            DataOut.writeBoolean ( true ) ;
            CKey   null ,
            Application   0 ;
```

```
        }
        else
        {
            // Keys do not match (abnormal shutdown)
            DataOut.writeBoolean ( false ) ,
            throw new IOException () ;
        }
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  GetApplication
//
//  Arguments:
//      none
//
//  Returns a byte buffer that contains all the
//  information necessary to run the application
//  and the application itself.
//
//  Sends TextEditor application
//
private byte []
GetApplication ()
{
    try
    {
        ByteArrayOutputStream Byter   new ByteArrayOutputStream () ;
        DataOutputStream IODev   new DataOutputStream ( Byter ) ,

        IODev.writeInt ( 2 ) ; // Elements in execuatable name

        IODev.writeUTF ( "java" ) ; // Executable Name
        IODev.writeUTF ( "Text_Editor" ) ; // Startup Class Name

        IODev.writeInt ( 1 ) ; // Number of files

        IODev.writeUTF ( "Text_Editor.class" ) , // File Name

        FileInputStream InFile   new FileInputStream ( "Text_Editor.class" ) ,

        int Length   InFile.available ( ) ,

        byte FileBuffer []   new byte [Length] ;

        InFile.read ( FileBuffer, 0, FileBuffer.length ) ; // Read File to buffer here

        InFile.close () ;

        IODev.writeInt ( FileBuffer.length ) ;  // Send file length out
        IODev.write ( FileBuffer, 0, FileBuffer.length ) , // Send file out

        return Byter.toByteArray() ,
    }
    catch ( IOException e )
    {
        return null ,
    }
}

////////////////////////////////////////////
//  GetFileManager
//
//  Arguments:
```

```
//      none
//
//  Returns a byte buffer that contains all the
//  information necessary to run the application
//  and the application itself.
//
//  Sends File Manager application
//
private byte []
GetFileManager ()
{
    try
    {
        ByteArrayOutputStream Byter   new ByteArrayOutputStream () ,
        DataOutputStream IODev   new DataOutputStream ( Byter ) ,

        IODev.writeInt ( 2 ) ; // Elements in execuatable name

        IODev.writeUTF ( "java" ) ; // Executable Name
        IODev.writeUTF ( "FileMgr" ) , // Startup Class Name

        IODev.writeInt ( 9 ) ; // Number of files

        String nameList []   new String [9] ,

        nameList [0] = new String ( "FileMgr.class" ) ,
        nameList [1]   new String ( "FMAboutDialog.class" ) ;
        nameList [2] = new String ( "FMDirPropertiesDialog.class" ) ,
        nameList [3] = new String ( "FMFilePropertiesDialog.class" ) ,
        nameList [4]   new String ( "FMNewFolderDialog.class" ) ,
        nameList [5]   new String ( "FMOpenRemoteDialog.class" ) ;
        nameList [6]   new String ( "FMQuitDialog.class" ) ;
        nameList [7] = new String ( "FMRenameDialog.class" ) ;
        nameList [8]   new String ( "DirectoryListEntry.class" ) ;

        for ( int i   0 , i < 9 , i++ )
        {
            IODev.writeUTF ( nameList[i] ) ; // File Name

            FileInputStream InFile   new FileInputStream ( nameList[i] ) ;

            int Length   InFile.available ( ) ,

            byte FileBuffer []   new byte [Length] ;

            InFile.read ( FileBuffer, 0, FileBuffer.length ) , // Read File to buffer here

            InFile.close () ;

            IODev.writeInt ( FileBuffer.length ) ,  // Send file length out
            IODev.write ( FileBuffer, 0, FileBuffer.length ) , // Send file out
        }

        return Byter.toByteArray() ,
    }
    catch ( IOException e )
    {
        return null ;
    }
}

//////////////////////////////////////////////////////
//Public Member Variables
//
//  Stage Values -> Indicate what stage the current request
//          is working at.
public final static
int NEWSERVICE   1 ;
```

```
    public final static
    int GETAPPLICATION = 2 ,
    public final static
    int CLOSESERVICE   3 ,
}
```

ASCommManager

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    AS Communication Manager
*
*
*    Jeff Harman
*    Date        Revision    Comment
*    4/1/97      1.0         Creation
*
*   Constructor for ASComMananger is all that is necessary
*   to spawn the AS Server
*
*   Intended to accept new connections for AS Server and spawn client
*   threads to handle them (keeping the original port open)
*/

/**
*   Package: ASServer
*       Package intended to be repository of all AS Server object
*/
package ASServer ;

/**
*   System Import(s)
*/
import java.io.* ,
import java.net.* ;
/**
*   Custom Import(s)
*/
import Keys.* ;
import ASUtils.* ,

/**
*   Class Definition
*/
public
class ASCommManager
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // Local_Host -> Name of this machine
    private String
    Local_Host = null ,

    ////////////////////////////////////////////
    //  Constructor
    //
```

```
//  Arguments:
//      none
//
//  Creates all necessary data management objects to run the server
//  Creates and infinte loop to block on waiting for new client
//          connections.  Spawns a new client to handle each new
//          client connection.
//
public
ASCommManager ()
{
    try
    {
        // Set up connection socket
        ServerSocket Key_Server   new ServerSocket ( AS_API.Well_Known_Port ) , ,
        Socket ReplySocket   null ;

        // Get the machine name
        InetAddress LHost   InetAddress.getLocalHost () ,
        Local_Host   LHost.toString () ,

        int Position   Local_Host.indexOf( '/' ) ;
        Local_Host   new String ( Local_Host.substring(Position+1) ) ,

        // Set up output
        System.out.println("Application Server:") ,
        System.out.println("-----------------------------") ;
        System.out.println( "Machine Name: "+Local_Host ) ,
        System.out.println( "-----------------------------") ,

        // Wait for new clients
        while ( true )
        {
            ReplySocket   Key_Server.accept () ,
            // Start a new thread to handle the clients
            NewService ( ReplySocket) ,
        }
    }
    catch ( UnknownHostException e )
    {
        System.out.println ("IO Error.\n") ;
    }
    catch ( IOException e )
    {
        System.out.println ("IO Error.\n") ;
    }
}

////////////////////////////////////////////
//  NewService
//
//  Arguments:
//      cSocket -> Client connection socket
//
//  Spawn a thread to handle the new client
//
private void
NewService ( Socket cSocket )
{
    System.out.println ("New Client") ,
    ASClient ClientThread   new ASClient ( cSocket ) ;
    // next two lines are probably not necessary
    // Thread CThread = new Thread ( ClientThread ) ,
    // CThread.start () ,

    // Start the thread
    ClientThread.start () ,
}
```

```
}
```

ASServer

```
/*****************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITHESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*****************************************************************************/

/*****************************************************************************
*    ASServer (& Main)
*
*    Spawn a new AS Server
*/

/**
*   Custom Imports
*/
import ASServer.* ,

/**
*   Class Definition
*/
public
class ASServer
{
    ////////////////////////////////////////////
    //  Constructor
    //
    //  Arguments:
    //      none
    //
    //  Spawn a new ASCommManager
    //  (and thereby start a new AS Server)
    public ASServer ()
    {
        new ASCommManager () ,
    }

    ////////////////////////////////////////////
    //  main
    //
    //  Arguments:
    //      args -> List of command line arguments (ignored)
    //
    //  Just create a new AS Server
    //
    public static void main ( String args [] )
    {
        new ASServer ( ) ,
    }
}
```

Keys

File List:
Channel_Key.java
Service_Key.java

Channel_Key

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    Channel Key Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    2/17/97     1.0         Creation
*
*    Key that is used by the system to maintain service requests.
*
*    Fairly simple data management tool, also can create and use
*    byte streams to allow easier transportation of information.
*
*/

/**
*   Package: Keys
*       Package intended to be repository of all Key (and related)objects
*/
package Keys ,

/**
*   System Imports
*/
import java.io.* ,

/**
*   Class Definition
*/
public
class Channel_Key
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  sUserName -> contains user identifier
    private String sUserName ;
    //  bKeyInfo -> Used to contain encrypted information
    //              about the user for server use
    private byte   bKeyInfo [] ;

    ////////////////////////////////////////////
    //  Constructor (Empty)
    //
    //  Arguments:
    //      none
    //
    //  Creates and empty key with null data fields
    //
```

```
public
Channel_Key ()
{
    sUserName   null ,
    bKeyInfo = null ;
}

////////////////////////////////////////////
//  Constructor (Stream)
//
//  Arguments:
//      InStream -> Data stream that contains the information to
//              construct the stream
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
public
Channel_Key ( byte InStream [] )
{
    ByteArrayInputStream Byter   new ByteArrayInputStream (InStream) ,
    DataInputStream IODev   new DataInputStream ( Byter ) ,

    try
    {
        sUserName   new String ( IODev.readUTF () ) ;
        bKeyInfo   new byte [IODev.readInt ()] ,
        for ( int i = 0 , i < bKeyInfo.length , i++ )
            bKeyInfo[i]= IODev.readByte () ;
    }

    catch ( IOException e )
    {
    }
}

////////////////////////////////////////////
//  Constructor (Copy)
//
//  Arguments:
//      Original -> Channel key that will be duplicated
//
//  takes another Channel key as it's argument for construction
//
public
Channel_Key ( Channel_Key Original )
{
    // Convert the old key to a data strem
    byte InStream []   Original.getKeyStream () ,
    ByteArrayInputStream Byter = new ByteArrayInputStream (InStream) ,
    DataInputStream IODev   new DataInputStream ( Byter ) ;

    try
    {
        // Copy Information to memeber variables
        sUserName = new String ( IODev.readUTF () ) ;
        bKeyInfo   new byte [IODev.readInt ()] ,
        for ( int i = 0 ; i < bKeyInfo.length , i++ )
            bKeyInfo[i]= IODev.readByte () ,
    }

    catch ( IOException e )
    {
    }
}

////////////////////////////////////////////
```

```
//  getUserName
//
//  Arguments:
//      none
//
//  Returns the user name stored in the key
//  Returns null if none defined
//
public String
getUserName ()
{
    String ReturnVal   null ;

    if ( sUserName != null )
        ReturnVal   new String ( sUserName ) ;

    return ReturnVal ,
}

////////////////////////////////////////////
//  setUserName
//
//  Arguments:
//      sName -> value to set the User name to
//
//  Sets the user name in the object
//
public void
setUserName ( String sName )
{
    sUserName   new String ( sName ) ,
}

////////////////////////////////////////////
//  getKey
//
//  Arguments:
//      none
//
//  Returns the byte key stored in the key
//  Returns null if none defined
//
public byte []
getKey ()
{
    byte ReturnVal []   null ,

    if ( bKeyInfo != null )
        ReturnVal   new byte [ bKeyInfo.length ] ,

    for ( int i   0 ; i < bKeyInfo.length ; i++ )
        ReturnVal[i]   bKeyInfo[i] ;

    return ReturnVal ,
}

////////////////////////////////////////////
//  setKey
//
//  Arguments:
//      bKey -> value to set the Byte Key to
//
//  Sets the Byte Key in the object
//
public void
setKey ( byte bKey[] )
{
    bKeyInfo   new byte [ bKey.length ] ;
```

```
    for ( int i   0 ; i < bKey.length , i++ )
        bKeyInfo[i] = bKey[i] ;
}

////////////////////////////////////////////
//  getKeyStream
//
//  Arguments:
//      none
//
//  Returns the entire Channel key encoded into
//  a byte buffer.
//
//  Returns null if any field has not been defined
//
public byte []
getKeyStream ()
{
    ByteArrayOutputStream Byter   new ByteArrayOutputStream () ,
    DataOutputStream IODev   new DataOutputStream ( Byter ) ,

    try
    {
        IODev.writeUTF ( sUserName ) ;
        IODev.writeInt ( bKeyInfo.length ) ;
        for ( int i   0 , i < bKeyInfo.length ; i++ )
            IODev.writeByte ( bKeyInfo[i] ) ;

        return Byter.toByteArray() ,
    }
    catch ( IOException e )
    {
        return null ,
    }

}

////////////////////////////////////////////
//  Compare
//
//  Arguments:
//      OtherKey -> Key to compare this key to
//
//  Returns true if the information contained in the
//      supplied key matches exactly
//
//  Returns false if there are any differences
//
public boolean
Compare ( Channel_Key OtherKey )
{
    if ( !OtherKey.getUserName().equals(sUserName) )
    {
        //System.out.println("User Name Fail") ,
        return false ;
    }

    byte OKeyFld[]   OtherKey.getKey() ,

    if ( OKeyFld.length != bKeyInfo.length )
    {
        //System.out.println("Byte Length Fail") ,
        return false ;
    }

    for ( int i = 0 , i < bKeyInfo.length ; i++ )
    {
```

```
            if ( OKeyFld[i] != bKeyInfo[i] )
            {
                //System.out.println("Ind Byte Compare Fail") ;
                return false ;
            }
        }

        return true ;
    }

    ////////////////////////////////////////////
    //  CompareKeys
    //
    //  Arguments:
    //      OtherKey -> Key to compare this key to
    //
    //  Returns true if the information contained in the
    //      supplied key matches exactly
    //
    //  Returns false if there are any differences
    //
    public boolean
    CompareKeys ( Channel_Key OtherKey )
    {
        if ( !OtherKey.getUserName().equals(sUserName) )
        {
            //System.out.println("User Name Fail") ;
            return false ,
        }

        byte OKeyFld[]   OtherKey.getKey() ;

        if ( OKeyFld.length != bKeyInfo.length )
        {
            //System.out.println("Byte Length Fail") ,
            return false ,
        }

        for ( int i   0 , i < bKeyInfo.length , i++ )
        {
            if ( OKeyFld[i] != bKeyInfo[i] )
            {
                //System.out.println("Ind Byte Compare Fail") ,
                return false ;
            }
        }

        return true ,
    }
}
```

Service_Key

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    Service Key Object
*
*    Key that is used by the system to create service requests.
*
*    Fairly simple data management tool, also can create and use
*    byte streams to allow easier transportation of information.
*
*    Jeff Harman
*    Date        Revision    Comment
*    2/15/97     1.0         Creation
*
*/

/**
*   Package: Keys
*       Package intended to be repository of all Key (and related)objects
*/
package Keys ,

/**
*   System Imports
*/
import java.io.* ;

/**
*   Class Definition
*/
public class Service_Key
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  sUserName -> contains user name identifier
    private String
    sUserName ;
    //  sUserHost -> contains user primary UNS server address
    private String
    sUserHost ;
    //  sServiceRequest -> contains service being requested
    private String
    sServiceRequest ;
    //  sAdditionalInfo -> contains any additional information
    private String
    sAdditionalInfo ,
    //                      needed by the server
    //  bKeyInfo -> Used to contain encrypted information
    private byte
```

```
bKeyInfo [] ,
//              about the user for server use
//  iPort -> The port that the user's primary UNS server
//           uses
private int
iPort ,

////////////////////////////////////////////
//  Constructor (Empty)
//
//  Arguments:
//      none
//
//  Creates and empty key with null data fields
//
public
Service_Key ()
{
    sUserName   null ;
    sUserHost   null ,
    sServiceRequest = null ,
    sAdditionalInfo   null ;
    bKeyInfo   null ,
    iPort = 0 ;
}

////////////////////////////////////////////
//  Constructor (Stream)
//
//  Arguments:
//      InStream -> Data stream that contains the information to
//              construct the stream
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
public
Service_Key ( byte InStream [] )
{
    ByteArrayInputStream Byter   new ByteArrayInputStream (InStream) ,
    DataInputStream IODev   new DataInputStream ( Byter ) ,

    try
    {
        sUserName   new String ( IODev.readUTF () ) ;
        sUserHost   new String ( IODev.readUTF () ) ;
        bKeyInfo   new byte [IODev.readInt ()] ,
        for ( int i   0 , i < bKeyInfo.length , i++ )
            bKeyInfo[i]= IODev.readByte () ;
        sServiceRequest = new String ( IODev.readUTF () ) ,
        sAdditionalInfo   new String ( IODev.readUTF () ) ,
        iPort   IODev.readInt () ;
    }

    catch ( IOException e )
    {
    }
}

////////////////////////////////////////////
//  Constructor (Copy)
//
//  Arguments:
//      Original -> Service key that will be duplicated
//
//  takes another service key as it's argument for construction
//
```

```
public
Service_Key ( Service_Key Original )
{
    byte InStream [] = Original.getKeyStream () ,
    ByteArrayInputStream Byter  new ByteArrayInputStream (InStream) ;
    DataInputStream IODev   new DataInputStream ( Byter ) ;

    try
    {
        sUserName = new String (IODev.readUTF () ) ,
        sUserHost = new String (IODev.readUTF () ) ,
        bKeyInfo   new byte [IODev.readInt ()] ;
        for ( int i = 0 , i < bKeyInfo.length , i++ )
            bKeyInfo[i]= IODev.readByte () ,
        sServiceRequest   new String (IODev.readUTF () ) ,
        sAdditionalInfo   new String (IODev.readUTF () ) ,
        iPort   IODev.readInt () ,
    }

    catch ( IOException e )
    {
    }
}

////////////////////////////////////////////
//  getUserName
//
//  Arguments:
//      none
//
//  Returns the user name stored in the key
//  Returns null if none defined
//
public String
getUserName ()
{
    String ReturnVal   null ,

    if ( sUserName != null )
        ReturnVal   new String ( sUserName ) ,

    return ReturnVal ;
}

////////////////////////////////////////////
//  getUserHost
//
//  Arguments:
//      none
//
//  Returns the user host stored in the key
//  Returns null if none defined
//
public String
getUserHost ()
{
    String ReturnVal   null ;

    if ( sUserHost != null )
        ReturnVal = new String ( sUserHost ) ;

    return ReturnVal ,
}

////////////////////////////////////////////
//  getServiceRequest
//
//  Arguments:
```

```
//      none
//
//  Returns the Servcie Request stored in the key
//  Returns null if none defined
//
public String
getServiceRequest ()
{
    String ReturnVal   null ,

    if ( sServiceRequest != null )
        ReturnVal   new String ( sServiceRequest ) ;

    return ReturnVal ,
}

////////////////////////////////////////////
//  getAdditionalInfo
//
//  Arguments:
//      none
//
//  Returns the Additional Information stored in the key
//  Returns null if none defined
//
public String
getAdditionalInfo ()
{
    String ReturnVal = null ;

    if ( sAdditionalInfo != null )
        ReturnVal = new String ( sAdditionalInfo ) ;

    return ReturnVal ;
}

////////////////////////////////////////////
//  getKey
//
//  Arguments:
//      none
//
//  Returns the Byte Key stored in the key
//  Returns null if none defined
//
public byte []
getKey ()
{
    byte ReturnVal []   null ,

    if ( bKeyInfo != null )
        ReturnVal   new byte [ bKeyInfo.length ] ;

    for ( int i   0 , i < bKeyInfo.length , i++ )
        ReturnVal[i]   bKeyInfo[i] ;

    return ReturnVal ,
}

////////////////////////////////////////////
//  getHostPort
//
//  Arguments:
//      none
//
//  Returns the Host Port stored in the key
//  Returns null if none defined
//
```

```
public int
getHostPort ( )
{
    return iPort ,
}

////////////////////////////////////////////
//  getKeyStream
//
//  Arguments:
//      none
//
//  Returns the Service Key encoded into a byte array
//  Returns null if any of the fields are undefined
//
public byte []
getKeyStream ()
{
    ByteArrayOutputStream Byter   new ByteArrayOutputStream () ,
    DataOutputStream IODev   new DataOutputStream ( Byter ) ,

    try
    {
        IODev.writeUTF ( sUserName ) ,
        IODev.writeUTF ( sUserHost ) ,
        IODev.writeInt ( bKeyInfo.length ) ;
        for ( int i   0 ; i < bKeyInfo.length , i++ )
            IODev.writeByte ( bKeyInfo[i] ) ,
        IODev.writeUTF ( sServiceRequest ) ,
        IODev.writeUTF ( sAdditionalInfo ) ;
        IODev.writeInt ( iPort ) ;

        return Byter.toByteArray() ,
    }
    catch ( IOException e )
    {
        return null ,
    }

}

////////////////////////////////////////////
//  setUserName
//
//  Arguments:
//      sName -> The value to set the user name to
//
//  Sets the user name to the value specified in sName
//
public void
setUserName ( String sName )
{
    sUserName   new String ( sName ) ,
}

////////////////////////////////////////////
//  setUserHost
//
//  Arguments:
//      sHost -> The value to set the user host to
//
//  Sets the user host to the value specified in sHost
//
public void
setUserHost ( String sHost )
{
    sUserHost   new String ( sHost ) ;
}
```

```
////////////////////////////////////////////
//  setServiceRequest
//
//  Arguments:
//      sRequest -> The value to set the Service Request to
//
//  Sets the Service Request to the value specified in sRequest
//
public void
setServiceRequest ( String sRequest )
{
    sServiceRequest   new String ( sRequest ) ,
}

////////////////////////////////////////////
//  setAdditionalInfo
//
//  Arguments:
//      sInfo -> The value to set the Additional Information to
//
//  Sets the Additional Information to the value specified in sInfo
//
public void
setAdditionalInfo ( String sInfo )
{
    sAdditionalInfo   new String ( sInfo ) ;
}

////////////////////////////////////////////
//  setAdditionalInfo
//
//  Arguments:
//      bKey -> The value to set the Byte Key to
//
//  Sets the Byte Key to the value specified in bKey
//
public void
setKey ( byte bKey[] )
{
    bKeyInfo   new byte [ bKey.length ] ,

    for ( int i = 0 , i < bKey.length ; i++ )
        bKeyInfo[i]   bKey[i] ;
}

////////////////////////////////////////////
//  setAdditionalInfo
//
//  Arguments:
//      Port -> The value to set the Host Port to
//
//  Sets the Host Port to the value specified in Port
//
public void
setHostPort ( int Port )
{
    iPort = Port ,
}

////////////////////////////////////////////
//  CompareKeys (Stream)
//
//  Arguments:
//      BStream -> Data stream that contains theKey
//                 to compare this key to
//
//  Returns true if the information contained in the
```

```
//      supplied key matches exactly
//
//  Returns false if there are any differences
//
public
boolean
CompareKeys ( byte BStream [] )
{
    Service_Key Test   new Service_Key ( BStream ) ,
    return CompareKeys ( Test ) ;
}

////////////////////////////////////////////
//  Compare (Copy)
//
//  Arguments:
//      OtherKey -> Key to compare this key to
//
//  Returns true if the information contained in the
//      supplied key matches exactly
//
//  Returns false if there are any differences
//
public
boolean
CompareKeys ( Service_Key OtherKey )
{
    if ( !OtherKey.getUserName().equals(sUserName) )
    {
        //System.out.println("User Name Fail") ,
        return false ;
    }

    if ( !OtherKey.getUserHost().equals(sUserHost) )
    {
        //System.out.println("User Host Fail") ,
        return false ,
    }

    if ( OtherKey.getHostPort() != iPort )
    {
        //System.out.println("Host Port Fail") ,
        return false ,
    }

    if ( !OtherKey.getServiceRequest().equals(sServiceRequest) )
    {
        //System.out.println("Serv Req Fail") ;
        return false ,
    }

    if ( !OtherKey.getAdditionalInfo().equals(sAdditionalInfo) )
    {
        //System.out.println("Addit Info Fail") ,
        return false ;
    }

    byte OKeyFld[]   OtherKey.getKey() ,

    if ( OKeyFld.length != bKeyInfo.length )
    {
        //System.out.println("Byte Length Fail") ,
        return false ;
    }

    for ( int i   0 , i < bKeyInfo.length , i++ )
    {
        if ( OKeyFld[i] != bKeyInfo[i] )
```

```
            {
                //System.out.println("Ind Byte Compare Fail") ,
                return false ;
            }
        }

        return true ;
    }
}
```

Utility

File List:
Heirarchy_Manager .java
Config_Manager.java
User_Info_Manager.java

Heirarchy_Manager

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    Heirarchy Manager Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    3/25/97     1.0         Creation
*
*   Manages a list of service keys.
*   Allows access, transportation, modification and
*   retrevial to and from the list.
*/

/**
*   Package: UNSUtils
*       Package intended to be repository of all UNS Server interface
*       and utility objects
*/
package UNSUtils ;

/**
*   System Imports
*/
import java.io.* ;
import java.util.* ,
/**
*   Custom Imports
*/
import Keys.* ,

/**
*   Class Definition
*/
public
class Heirarchy_Manager
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  KeyList -> Vector that stores the key list
    private
    Vector KeyList ;

    ////////////////////////////////////////////
    //  Constructor (Empty)
    //
    //  Arguments:
    //      none
    //
```

```
//  Creates an empty heirarchy manager
public
Heirarchy_Manager ()
{
    KeyList   new Vector () ,
}

////////////////////////////////////////////
//  Constructor (Byte Stream)
//
//  Arguments:
//      InStream -> Data stream that contains the information to
//              construct the object (byte array)
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
public
Heirarchy_Manager ( byte InStream [] ) throws IOException
{
    try
    {
        ByteArrayInputStream Byter   new ByteArrayInputStream (InStream) ;
        DataInputStream Input   new DataInputStream( Byter ) ;
        StreamConstructor ( Input ) ;
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  Constructor (Copy)
//
//  Arguments:
//      Old_Heir -> Heirarchy Manager that will be duplicated
//
//  takes another Heirarchy Manager as it's argument for construction
//
public
Heirarchy_Manager ( Heirarchy_Manager Old_Heir ) throws IOException
{
    try
    {
        ByteArrayInputStream Byter
              new ByteArrayInputStream (Old_Heir.getDataStream() ) ;
        DataInputStream Input = new DataInputStream( Byter ) ,
        StreamConstructor ( Input ) ,
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  StreamConstructor
//
//  Arguments:
//      InputStream -> Data stream that contains the information to
//              construct the object
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
```

```
private void
StreamConstructor ( DataInputStream InputStream ) throws IOException
{
    try
    {
        int Key_Count   InputStream.readInt ( ) ;

        KeyList   new Vector ( Key_Count ) ,

        for ( int i   0 ; i < Key_Count , i++ )
        {
            int ArrayLength   InputStream.readInt() ;

            byte InputArray []   new byte [ArrayLength] ;

            for ( int j   0 ; j < InputArray.length , j++ )
            {
                InputArray[j] = InputStream.readByte () ,
            }

            Service_Key NewKey   new Service_Key( InputArray ) ,

            AddKey( NewKey ) ,
        }
    }

    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  Constructor (Stream)
//
//  Arguments:
//      InputStream -> Data stream that contains the information to
//              construct the object
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
public
Heirarchy_Manager ( DataInputStream InputStream ) throws IOException
{
    try
    {
        StreamConstructor ( InputStream ) ;
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  WriteHeirarchy
//
//  Arguments:
//      OutputStream -> Data stream that will receive the info
//          contained in the object
//
//  requires a vaild output stream to be passed in
//
public void
WriteHeirarchy ( DataOutputStream OutputStream ) throws IOException
{
```

```
    try
    {
        int Key_Count   GetTotalIndexs () ;

        OutputStream.writeInt ( Key_Count ) ,

        for ( int i   0 , i < Key_Count ; i++ )
        {
            Service_Key CurrentKey
                GetKeyAtIndex ( i + GetKeyIndex( (Service_Key) KeyList.firstElement () ) ) ,

            byte OutputArray []   CurrentKey.getKeyStream () ;

            OutputStream.writeInt ( OutputArray.length ) ,
            OutputStream.write ( OutputArray, 0, OutputArray.length ) ;
        }
    }

    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  getDataStream
//
//  Arguments:
//      none
//
//  requires a byte array that contains the
//  infromation contained in the object
//
public byte []
getDataStream () throws IOException
{
    ByteArrayOutputStream Byter   new ByteArrayOutputStream () ,
    DataOutputStream Output   new DataOutputStream( Byter ) ;
    WriteHeirarchy ( Output ) ,
    return Byter.toByteArray() ,
}

////////////////////////////////////////////
//  InsertAtIndex
//
//  Arguments:
//      NewKey -> Key to be inserted
//      Index -> Where the key will be inserted
//
//  Replaces the service key at the specified index.
public void
InsertAtIndex ( Service_Key NewKey, int Index )
{
    KeyList.setElementAt ( NewKey, Index ) ,
}

////////////////////////////////////////////
//  AddKey
//
//  Arguments:
//      NewKey -> Key to be inserted
//
//  Add a key to the end of the list
//
public void
AddKey ( Service_Key NewKey )
{
    KeyList.addElement(NewKey) ;
```

```
}

////////////////////////////////////////////
//  GetKeyIndex
//
//  Arguments:
//      NewKey -> Key to be found
//
//  find a key in the list
//  returns value of index or -1 if not found
//
public int
GetKeyIndex ( Service_Key NewKey )
{
    int i   0 ,
    boolean found   false ;

    while (( ! found ) && ( i < KeyList.size() ))
    {
        if ( GetKeyAtIndex ( i ).CompareKeys ( NewKey ) )
        {
            found   true ,
        }
        else
        {
            i++ ,
        }
    }

    if ( i < KeyList.size() )
        return i ;
    else
        return -1 ;
}

////////////////////////////////////////////
//  DeleteKeyAtIndex
//
//  Arguments:
//      Index -> Index of key to be deleted
//
//  deletes a key from the list at the specified index
//
public void
DeleteKeyAtIndex ( int Index )
{
    KeyList.removeElementAt( Index ) ;
}

////////////////////////////////////////////
//  GetKeyAtIndex
//
//  Arguments:
//      Index -> Index of key to be retreived
//
//  retreives a key from the list at the specified index
//
public Service_Key
GetKeyAtIndex ( int Index )
{
    Service_Key KeyFromList   (Service_Key) KeyList.elementAt ( Index ) ,

    return KeyFromList ;
}

////////////////////////////////////////////
//  GetTotalIndexs
//
```

```
    //  Arguments:
    //      none
    //
    //  Returns the total number of keys stored in the system
    //
    public int
    GetTotalIndexs ()
    {
        return KeyList.size() ,
    }

    ////////////////////////////////////////////
    //  IsInList
    //
    //  Arguments:
    //      CheckKey -> Key to be found in list
    //
    //  Determines if a key is in the list
    //
    //  returns true is key is in list
    //  returns false if it is not
    //
    public boolean
    IsInList ( Service_Key CheckKey )
    {
        boolean Response   false ,

        for ( int i = 0 ; i < KeyList.size() , i++ )
        {
            if ( CheckKey.CompareKeys( GetKeyAtIndex(i) ) )
            {
                Response   true ;
                break ,
            }
        }

        return Response ,
    }
}
```

Config_Manager

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    Configuration Manager Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    3/25/97     1.0         Creation
*
*   Stores and manages the configuration information for a user
*
*   currently only stores two color values.
*/

/**
*   Package: UNSUtils
*       Package intended to be repository of all UNS Server interface
*       and utility objects
*/
package UNSUtils ;

/**
*   System Imports
*/
import java.io.* ,
import java.awt.* ;

/**
*   Class Definition
*/
public
class Config_Manager
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  ForeColor -> Color of the user's foreground
    private
    Color ForeColor ,
    //  BackColor -> Color of the user's background
    private
    Color BackColor ;

    ////////////////////////////////////////////////
    //  Constructor (Empty)
    //
    //  Arguments:
    //      none
    //
    //  Creates an manager with the colors set to
```

```
//  the default values:
//      Foreground -> Black
//      Background -> White
//
public
Config_Manager ()
{
    ForeColor = new Color ( Color.black.getRed (),
                            Color.black.getGreen (),
                            Color.black.getBlue () ) ,
    BackColor = new Color ( Color.wbite.getRed (),
                            Color.white.getGreen (),
                            Color.wbite.getBlue () ) ;
}

////////////////////////////////////////////
//  Constructor (Stream)
//
//  Arguments:
//      InputStream -> Data stream tbat contains the information to
//              construct the object
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract tbe information from the byte stream and store it appropriatly.
//
public
Config_Manager ( DataInputStream InputStream ) throws IOException
{
    try
    {
        StreamConstructor ( InputStream ) ,
    }
    catch ( IOException e )
    {
        throw e ;
    }
}

////////////////////////////////////////////
//  Constructor (Byte Stream)
//
//  Arguments:
//      InStream -> Data stream that contains the information to
//              construct the object (byte array)
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
public
Config_Manager ( byte InStream [] ) throws IOException
{
    try
    {
        ByteArrayInputStream Byter   new ByteArrayInputStream (InStream) ;
        DataInputStream Input = new DataInputStream( Byter ) ,
        StreamConstructor ( Input ) ,
    }
    catch ( IOException e )
    {
        throw e ;
    }
}

////////////////////////////////////////////
//  Constructor (Copy)
//
```

```
//  Arguments:
//      OldConfig -> Configuration Manager that will be duplicated
//
//  takes another Config. Manager as it's argument for construction
//
public
Config_Manager ( Config_Manager OldConfig )
{
    ForeColor   new Color ( OldConfig.GetForegroundColor().getRed(),
                            OldConfig.GetForegroundColor().getGreen(),
                            OldConfig.GetForegroundColor().getBlue() ) ,
    BackColor   new Color ( OldConfig.GetBackgroundColor().getRed(),
                            OldConfig.GetBackgroundColor().getGreen(),
                            OldConfig.GetBackgroundColor().getBlue() ) ,
}

////////////////////////////////////////////
//  StreamConstructor
//
//  Arguments:
//      InputStream -> Data stream that contains the information to
//              construct the object
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
private boolean
StreamConstructor ( DataInputStream InputStream ) throws IOException
{
    boolean Response   false ;

    try
    {
        ForeColor   new Color ( InputStream.readInt (),
                                InputStream.readInt (),
                                InputStream.readInt () ) ,

        BackColor = new Color ( InputStream.readInt (),
                                InputStream.readInt (),
                                InputStream.readInt () ) ,
        Response   true ;
    }
    catch ( IOException e )
    {
        throw e ;
    }

    return Response ,
}

////////////////////////////////////////////
//  WriteStream
//
//  Arguments:
//      OutputStream -> Data stream that will receive the info
//          contained in the object
//
//  requires a vaild output stream to be passed in
//
public void
WriteStream (DataOutputStream OutputStream) throws IOException
{
    try
    {
        OutputStream.writeInt ( ForeColor.getRed () ) ;
        OutputStream.writeInt ( ForeColor.getGreen () ) ;
        OutputStream.writeInt ( ForeColor.getBlue () ) ,
```

```
        OutputStream.writeInt ( BackColor.getRed () ) ;
        OutputStream.writeInt ( BackColor.getGreen () ) ,
        OutputStream.writeInt ( BackColor.getBlue () ) ,
    }
    catch ( IOException e )
    {
        throw e ;
    }
}

////////////////////////////////////////////
//  getDataStream
//
//  Arguments:
//      none
//
//  requires a byte array that contains the
//  infromation contained in the object
//
public byte []
getDataStream () throws IOException
{
    try
    {
        ByteArrayOutputStream Byter   new ByteArrayOutputStream () ;
        DataOutputStream Output = new DataOutputStream( Byter ) ,
        WriteStream ( Output ) ,
        return Byter.toByteArray() ;
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  SetForegroundColor
//
//  Arguments:
//      NewFore -> Color to be set for foreground
//
//  requires a vaild color
//
public void
SetForegroundColor ( Color NewFore )
{
    ForeColor   new Color ( NewFore.getRGB () ) ,
}

////////////////////////////////////////////
//  SetBackgroundColor
//
//  Arguments:
//      NewBack -> Color to be set for background
//
//  requires a vaild color
//
public void
SetBackgroundColor ( Color NewBack )
{
    BackColor   new Color ( NewBack.getRGB () ) ;
}

////////////////////////////////////////////
//  GetForegroundColor
//
//  Arguments:
//      none
```

```
    //
    //  returns the current foreground color
    //
    public Color
    GetForegroundColor ( )
    {
        Color NewColor = new Color ( ForeColor.getRGB () ) ,

        return NewColor ;
    }

    ////////////////////////////////////////////
    //  GetBackgroundColor
    //
    //  Arguments:
    //      none
    //
    //  returns the current background color
    //
    public Color
    GetBackgroundColor ( )
    {
        Color NewColor   new Color ( BackColor.getRGB () ) ;

        return NewColor ;
    }
}
```

User_Info_Manager

```
/**************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. TNE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
**************************************************************************/

/**************************************************************************
*    User Information Manager Object
*
*    Jeff Harman
*    Date        Revision    Comment
*    3/25/97     1.0         Creation
*
*   Manages the user name and password for a UNS server.
*
*   Allows comparison, access (in and out) to the user name
*   Allows comparison, access (in only) to the user password
*/

/**
*   Package: UNSUtils
*       Package intended to be repository of all UNS Server interface
*       and utility objects
*/
package UNSUtils ;

/**
*   System Imports
*/
import java.io.* ,

/**
*   Class Definition
*/
public
class User_Info_Manager
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    //  User_Name -> User's name for comparison
    private
    String  User_Name ;
    //  User_Password -> User's password for comparison
    private
    String  User_Password ,

    ////////////////////////////////////////////
    //  Constructor (Empty)
    //
    //  Arguments:
    //      none
    //
    //  Creates an manager with the name and password set to
```

```
//  the default values:
//      name -> "John Doe"
//      password -> "pass"
//
public
User_Info_Manager ()
{
    User_Name = new String ( "John Doe" ) ,
    User_Password = new String ( "pass" ) ;
}

////////////////////////////////////////////
//  Constructor (Copy)
//
//  Arguments:
//      oldInfo -> User information to be copied
//
//  copies another Info Manager into itself
//
public
User_Info_Manager ( User_Info_Manager oldInfo )
{
     User_Name   new String ( oldInfo.GetName() ) ;
     User_Password = new String ( oldInfo.GetPass() ) ;
}

////////////////////////////////////////////
//  Constructor (Stream)
//
//  Arguments:
//      InputStream -> Data stream that contains the information to
//              construct the object
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
public
User_Info_Manager ( DataInputStream InputStream ) throws IOException
{
    try
    {
        StreamConstructor ( InputStream ) ,
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  Constructor (Byte Stream)
//
//  Arguments:
//      InStream -> Data stream that contains the information to
//              construct the object (byte array)
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
public
User_Info_Manager ( byte InStream [] ) throws IOException
{
    try
    {
        ByteArrayInputStream Byter   new ByteArrayInputStream (InStream) ;
        DataInputStream IODev   new DataInputStream ( Byter ) ;
```

```
            StreamConstructor ( IODev ) ;
        }
        catch ( IOException e )
        {
            throw e ,
        }
}

//////////////////////////////////////////
//  StreamConstructor
//
//  Arguments:
//      InputStream -> Data stream that contains the information to
//              construct the object
//
//  Byte stream constructor:
//      requires a vaild byte stream to be passed in, constructor will
//      extract the information from the byte stream and store it appropriatly.
//
private void
StreamConstructor ( DataInputStream IODev ) throws IOException
{
    try
    {
        User_Name   new String ( IODev.readUTF () ) ;
        User_Password   new String ( IODev.readUTF () ) ;
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

//////////////////////////////////////////
//  WriteStream
//
//  Arguments:
//      OutputStream -> Data stream that will receive the info
//          contained in the object
//
//  requires a vaild output stream to be passed in
//
public void
WriteStream (DataOutputStream IODev) throws IOException
{
    try
    {
        IODev.writeUTF ( User_Name ) ;
        IODev.writeUTF ( User_Password ) ,
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

//////////////////////////////////////////
//  getDataStream
//
//  Arguments:
//      none
//
//  requires a byte array that contains the
//  information contained in the object
public byte []
getDataStream () throws IOException
{
    try
```

```
    {
        ByteArrayOutputStream Byter   new ByteArrayOutputStream () ,
        DataOutputStream IODev = new DataOutputStream ( Byter ) ;
        WriteStream ( IODev ) ,
        return Byter.toByteArray() ,
    }
    catch ( IOException e )
    {
        throw e ,
    }
}

////////////////////////////////////////////
//  CheckName
//
//  Arguments:
//      InName -> name to be compared
//
//  Compare the input name to the stored name
//
public boolean
CheckName ( String InName )
{
    return ( 0 == User_Name.compareTo( InName )) ,
}

////////////////////////////////////////////
//  CheckPassword
//
//  Arguments:
//      InPassword -> Password to be compared
//
//  Compare the input Password to the stored Password
//
public boolean
CheckPassword ( String InPassword )
{
    return ( 0 == User_Password.compareTo( InPassword )) ;
}

////////////////////////////////////////////
//  SetName
//
//  Arguments:
//      InName -> Name to be stored
//
//  set the user name to the input value
//
public void
SetName ( String InName )
{
    User_Name = new String ( InName ) ,
}

////////////////////////////////////////////
//  SetPassword
//
//  Arguments:
//      SetPassword -> Password to be stored
//
//  set the user Password to the input value
//
public void
SetPassword ( String SetPassword )
{
    User_Password = new String ( InPassword ) ;
}
```

```
    ////////////////////////////////////////////
    //  GetName
    //
    //  Arguments:
    //      none
    //
    //  returns the currently stored user name
    //
    public String
    GetName ()
    {
        return new String ( User_Name ) ,
    }

    ////////////////////////////////////////////
    //  GetPass
    //
    //  Arguments:
    //      none
    //
    //  returns the currently stored user password
    private String
    GetPass ()
    {
        return new String ( User_Password ) ;
    }

    ////////////////////////////////////////////
    //  Compare
    //
    //  Arguments:
    //      OtherUser -> User info manager to be compared to
    //
    //  Compares this user manager to another manager
    //  returns true is stored values are equal
    //  else it returns false
    //
    public boolean
    Compare ( User_Info_Manager OtherUser )
    {
        if ( OtherUser.CheckName ( User_Name ) &&
             OtherUser.CheckPassword ( User_Password ) )
            return true ,
        else
            return false ,
    }
}
```

Text Editor

File List:
Text_Editor.java

Text_Editor

```
/*************************************************************************
* Copyright (c) 1997 Jeffrey I. Harman, All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL purposes and without
* fee is hereby granted provided that this copyright notice
* appears in all copies.
*
* THE AUTHOR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY
* OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHOR SHALL NOT BE LIABLE
* FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*************************************************************************/

/*************************************************************************
*    Text Editor Object (&Main)
*
*        Simple application, intended to demonstrate the
*        application spawning and communication interface
*        of the Desktop.
*
*        Creates a large text pane that the user can input
*        text and clear it.
*
*    Jeff Harman
*    Date        Revision    Comment
*    4/4/97      1.0         Creation
*    4/5/97      1.1         Heirarchy_Manager Additions
*    4/12/97     1.2         EXE_API added
*********/

/**
*   System Import(s)
*/
import java.awt.* ,
import java.io.* ;
import java.util.* ,
/**
*   Custom Import(s)
*/
import DesktopUtils.* ,

/**
*   Class Definition
*/
public
class Text_Editor extends Frame
{
    ////////////////////////////////////////////////////
    //Private Member Variables
    //
    // TextAr -> Text Area for user editing
    private
    TextArea TextAr   null ,
    //
    // UButton -> User Button (Clear text pane)
    private
    Button UButton = null ,
    //
    // SButton -> System Button (Close application)
    private
    Button SButton   null ,
```

```
//////////////////////////////////////////
//  Constructor
//
//  Arguments:
//      none
//
//  Builds primary window and initiates
//      communication with the desktop.
public
Text_Editor ()
{
    super ( "Basic Text Editor" ) ; // Build window

    setLayout( new BorderLayout () ) , // Set layout and Panels
    Panel top   new Panel () ;
    Panel bottom   new Panel () ;
    top.setLayout ( new BorderLayout () ) ,
    bottom.setLayout ( new FlowLayout () ) ;

    TextAr   new TextArea("Enter your text here.\n", 15, 5) ;
    TextAr.setEditable ( true ) , // Build text area and set it's
    top.add ("Center", TextAr ) ; // attributes

    UButton   new Button("Clear") ;
    SButton   new Button("Close") ;
    bottom.add( UButton ) ,  // Build buttons
    bottom.add( SButton ) ,

    add("Center", top ) ;  // Add Panels to layout
    add("South", bottom ) ;

    resize(300, 300) ,  // Set pane size & attributes and display
    setResizable( false ) ;
    show () ;

    EXE_API TestAPI = new EXE_API () ;

    Vector Test1   TestAPI.GetHeirarchy () ,

    String Test2   TestAPI.GetUserFSSHost () ,

    int Test3   TestAPI.GetUserFSSPort () ,

    TestAPI.SendMessage ( "Server: "+Test2+" at port: "+Test3 ) ,

    TestAPI.SendMessage ( "Keys: "+Test1.size () ) ;
}

//////////////////////////////////////////
// handleEvent
//
//  Arguments:
//      evt -> System generated event object
//
//  Respond to system Events (button clicks etc..)
//
public boolean
handleEvent ( Event evt )
{
    // User Button (Clear Screen)
    if ( evt.target.equals ( UButton ) )
    {
        //Clear
        TextAr.setText ( "" ) ,
    }
    // System Button (Close Screen)
    else if ( evt.target.equals ( SButton ) )
```

```
        {
            //Close
            System.exit ( 0 ) ;
            return true ,
        }
        // System Window Close
        else if ( evt.id == Event.WINDOW_DESTROY )
        {
            //Close
            System.exit ( 0 ) ;
            return true ,
        }
        // We didn't handle it to pass it up
        return super.handleEvent( evt ) ;
    }

    ////////////////////////////////////////////
    // main
    //
    //  Arguments:
    //      args -> array of strings (that is command line)   ignored
    //
    //  Start the system and make a new instance of
    //  Text editor
    //
    public static void
    main ( String args [] )
    {
        new Text_Editor ( ) ,
    }
}
```


IMATION
3M
ENHANCED PERFORMANCE
FULL THESIS, ZIP
SOURCE CODE. ZIP
JEFF HARMAN
MAY 20, 1997
1.44 MB
High Density
Haute densite
IBM
Formatted
Formatee